



Drive Control

Enclustra GmbH, Oliver Bründler

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Example applications
 - Robotics
 - Industrial automation
 - Semiconductor
 - Medical testing
 - Automotive
- Focus of this presentation
 - Drive control (motion control)
 - Important concepts
 - Control of different motor types
 - Common mistakes

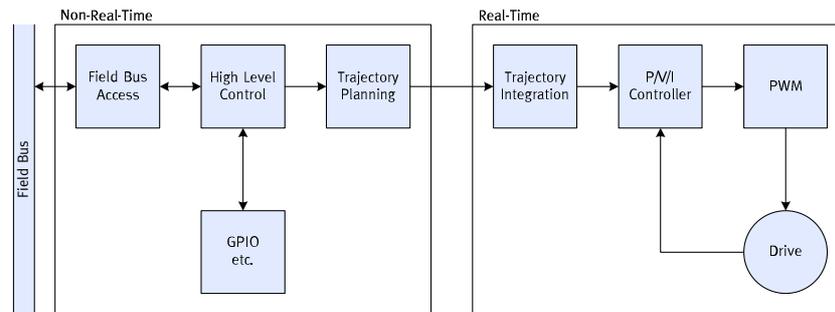


Electronic drive control systems exist for several years now and the requirements are increasing constantly. This leads to the requirement for more processing power and higher integration while cost should decrease.

Because integrating a drive control system into an FPGA can lead to substantial cost savings and enhanced performance, the usage of FPGA technology in this field of applications has become popular during the last few years.

This presentation focuses on motion control systems while a major part of the content does also apply for speed control systems. The main intention of this presentation is to give an overview of the most common system concepts and to help avoiding common mistakes.

- The generic drive control system



Most motion control systems are connected to a field bus in some way and high level control was often achieved by IPCs in the past. However, with the increased performance requirements and the availability of powerful low cost parts, the trend to distributed systems with a certain level of intelligence in its end points came up. A concept based on centralized intelligence often can just not achieve the high requirements of today's applications.

This high level control tasks lead to the requirement for executing non-real-time tasks in addition to the real-time tasks (trajectory integration, controller calculation, power electronics control) that used to be present for a while now.

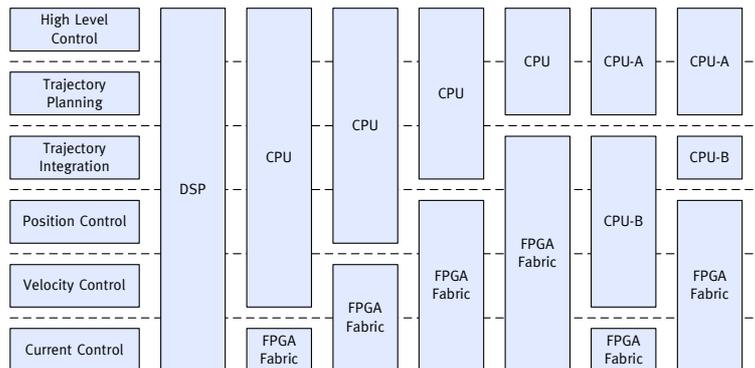
High level control may not only contain the field bus protocol stack and minor I/O functionality but also trajectory planning. The aim of this relatively complex task is to calculate a movement to a certain position without exceeding speed-, acceleration- and jerk-limits.

In the real-time domain, the desired values for the variables mentioned above are calculated constantly and controllers try to reach them.

To access today's most popular field buses (e.g. EtherCAT, Powerlink, SERCOS, CAN), specific circuitry is required. FPGAs can handle these tasks very well because IP cores for the most common field buses are available from different vendors and also as open source.

- Overview
- [Task Partitioning](#)
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Some kind of CPU is required for high level control (maybe with OS)
- There is no „golden“ system setup
- The best implementation depends on the requirements



Some time ago, the general drive control system contained one DSP that controlled everything. If this solution can satisfy all requirements, it is still worth to be considered.

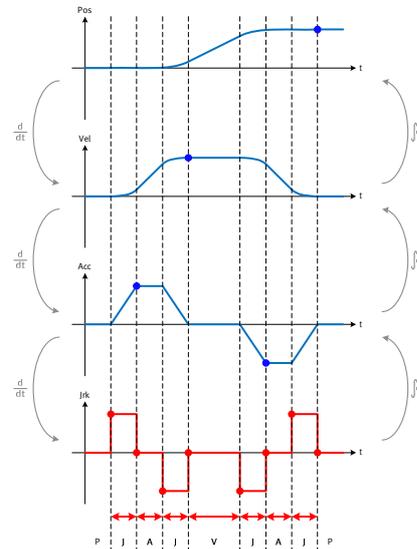
For more complex applications, a solution based on one single DSP often is out of question. So let's have a look at FPGA based approaches.

Because of the high level control tasks including trajectory planning, at least one CPU is required in (almost) every motion control system. Controllers can easily be realized in the FPGA fabric to increase the control rates. The same applies for simple trajectory integrators. But trajectory integrators can become complex if the requirements are not only of basic nature. Therefore a second CPU may also be a reasonable approach for trajectory integration, especially if floating point calculations are required.

These days, the usual choice for the CPUs are soft core processors because hardwired processors are not yet available in low-cost FPGAs. This is assumed to change in near future. Zynq from Xilinx for example offers two CPUs (and CAN interface as well as ADCs). It is obvious that this will fits exactly to the requirements of a typical motion control system.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- About trajectories



The information about any movement can be shown in different physical quantities that have a differential/integrational relationship to each other. It can for example be shown as position vs. time or as acceleration vs. time.

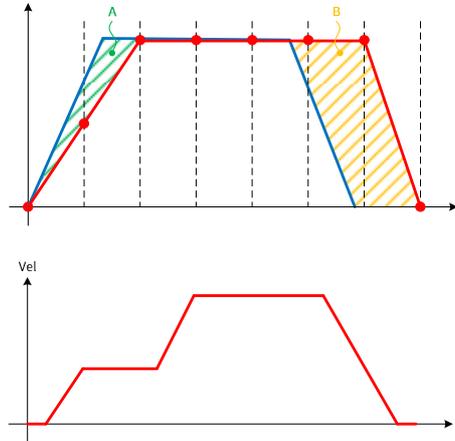
In most motion control systems velocity and acceleration are limited. Today it is state of the art to also limit the jerk, which leads to the well known s-curves. In some high-end applications even the derivative of the jerk is limited.

Theoretically the full information about a movement is contained in the highest derivative. In the figure above this leads to a dataset of 8 jerk/time pairs.

To get all quantities out of this dataset, it has to be integrated by the trajectory integrator. Since this integration has a finite precision, the results may be slightly wrong after a long time of integration. This can be solved by additionally passing end values for acceleration, velocity and position to the trajectory integrator.

Note that it is known for example that the jerk is zero during a constant acceleration phase. Therefore only 8 value/time pairs are required to describe the move above.

- Trajectory planning flow
 - Decide which type to use
 - JAJVJAJ
 - JAJJAJ
 - ...
 - Calculate phase durations
 - Round up phase durations
 - Find values to exactly reach the end position
 - Solve equation system
 - Exactly one solution
- Requirements
 - Many decisions to take -> CPU
 - FPU required



The first task when calculating a trajectory for a move is to decide what phase types it should contain. For simple moves with one constant velocity phase, there is a limited number of possibilities:

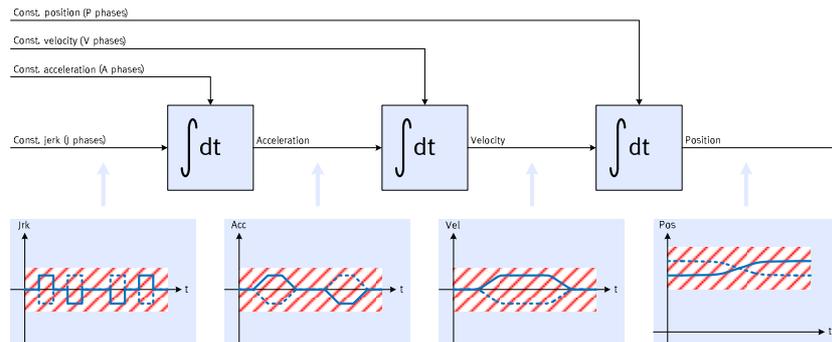
- JAJVJAJ: Acceleration limit and velocity limit can be reached
- JAJJAJ: The move is too short to reach the velocity limit but the acceleration limit can be reached
- JJVJJ: The velocity limit is too low to reach the acceleration limit but the velocity limit can be reached
- JJJ: The move is too short to reach the velocity and the acceleration limit

If the required phases are known, their duration has to be calculated. Afterwards all phase durations are rounded up to fit the discrete trajectory integrator cycles. Of course this rounding affects the end position. To prevent this position error, an equation system containing the total move duration, all limits (jerk, acceleration, velocity) and the end position has to be solved. For simple moves with one constant velocity phase this leads to exactly one solution.

Trajectory planning is a relatively computing intensive task and does require a floating point unit even for simple moves. For moves with higher requirements (e.g. more than one constant velocity phase) the effort to plan trajectories increases more than linearly.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Cascade of loadable integrators
- High dynamic ranges for jerk and acceleration
- High absolute values for position
- The crux of floating point: $10'000'000+1 = 10'000'000$



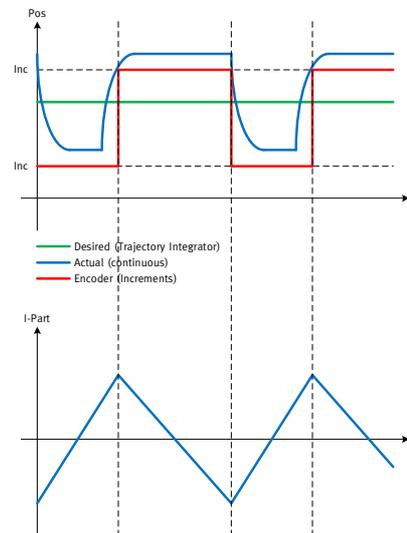
The trajectory integrator consists of three loadable integrators. This allows to calculate a position from a jerk. The integrators have to be loadable to make it possible to set the velocity integrator to a fixed velocity during the constant velocity phase for example.

For velocity, acceleration and jerk, floating point calculations can make things easier. Especially the jerk has a high dynamic range. Absolute values for these quantities are always around zero and the durations of these phases are normally not very long. Therefore floating point can be a reasonable choice, which normally leads to a SW implementation on a CPU with FPU. However, for low and intermediate requirements, floating point may not be required (this is usually the premise for a realization in FPGA fabric).

In contrast to the other values, the position can have very high absolute values while the change in position may be small. For long moves at a low speed this can lead to significant integration errors due to the truncation of floating point values. As a result of this, the position after such a move may vary from the desired position which results in a jump after switching back to position control after the execution of a movement.

This issue can only be solved by using fixed point or integer positions. The position unit may for example be increments of $1/32'768$ rotations. To calculate the position in floating point is only possible for systems with a small move range.

- Finite position resolution leads to ringing when standing still

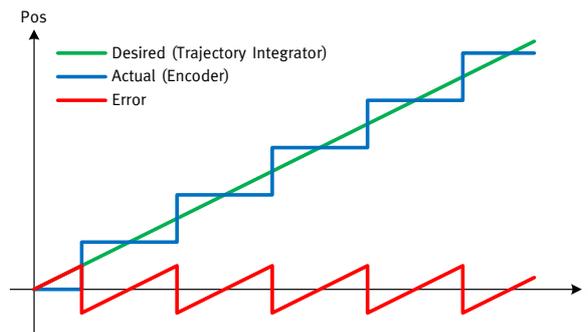


To achieve the required precision for long moves, the position resolution of the trajectory integrator is usually higher than the position resolution of the encoder.

If the desired position lies in between of two encoder increments, it can never be reached exactly. This leads to the following problem if the motor stands still: The integral part of the position controller increases until it is high enough to overcome the static friction. As a result of this, the motor turns a bit and passes the next increment. Because of the resolution of the encoder, the actual position does still differ from the desired position but in the reverse direction. Therefore the integral part of the controller begins to increase in the other direction and the drive always toggles +/- 1 encoder increment around the desired position.

This phenomenon is called "ringing" and can lead to noise even if the motor does not move. Other bad effects of this problem are vibrations that can affect lifetimes of motors and mechanics as well as unnecessary power consumption.

- Finite position resolution leads to noise during slow movements
- Solution: Proper rounding



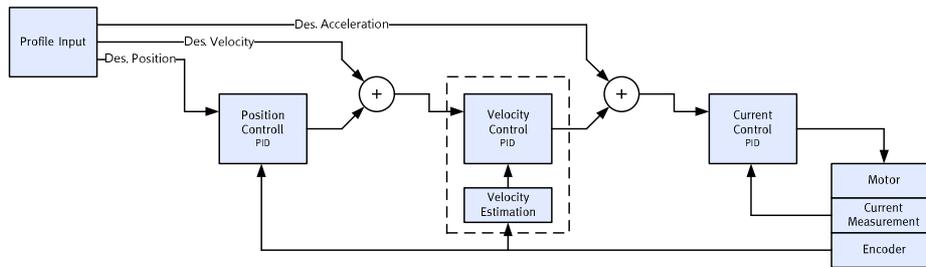
Another effect very similar to ringing occurs during slow movements:

The desired position from the trajectory integrator increases linearly or at least with a much smaller step size than the actual position from the encoder. This leads to a periodic deviation that is always smaller than ± 1 increment.

This deviation can cause clearly hearable noise because of its periodicity. This is especially true for systems with high derivative gain in the position controller.

For both issues, ringing and noise during slow movements, there is a solution: The output of the trajectory integrator has to be rounded to the same step size that is available from the encoder. As a result of this a position deviation of zero can be achieved in all situations.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions



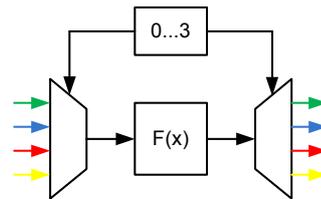
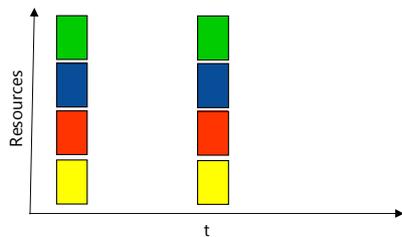
For DC/EC-motor based drive control systems, the most common controller architecture is a cascade of multiple PID controllers with feed forward paths.

Theoretically three controllers are used (one for every physical quantity: position, velocity, current/acceleration) while mid- and low-performance systems often get by without velocity controller. In this case, the velocity control is implicitly integrated in the position controller since the proportional part of the position controller is the same as the integral part of the velocity controller. The same applies for the derivative part of the position controller and the proportional part of the velocity controller.

From the control theory point of view, a correctly parameterized control loop looks like a low pass filter for the next higher order control loop. This allows to parameterize the different stages independently.

For stability reasons, every outer loop has to run at a lower speed than the inner loops. For example: 100kHz current control, 20kHz velocity control and 5 kHz position control.

- Concept of time division multiplexing
 - FPGAs are fast... (faster than required for motion control)
 - Use available processing power to minimize resource usage

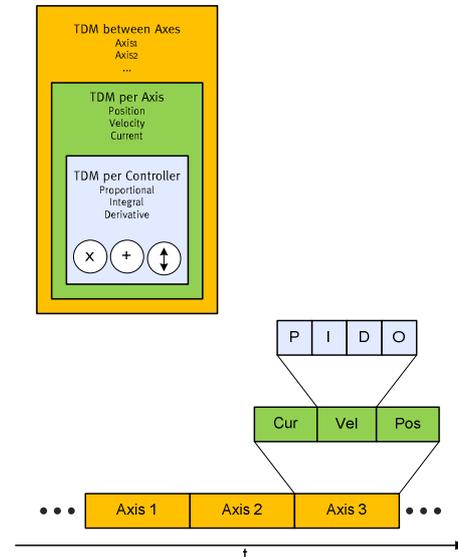


Today's drive control systems with maximum controller rates of about 100-200kHz for current control loops are relatively slow compared to what today's low-cost FPGAs are capable of. This leads to the usage of time division multiplexing (TDM) wherever possible.

TDM uses the same combinatorial logic to do identical calculations in a different context (e.g. for multiple axes). The calculations are executed one after the other in the same resources instead of being executed in parallel with each one using its own resources.

Efficient implementation

- Requirements
 - 3 controllers per axis
 - 3-5 multiplications per controller
 - Real world PIDs need also limiting, barrel shifter etc.
 - Price (chip size)
- Implementation
 - TDM on three levels
 - Only one multiplier and one adder
 - Control logic is reused on different levels
 - Well scalable to the number of axes
- Example: 8 axes, 200kHz current control, 80MHz clock
 - 8 axes x 200kSps x 3 PIDs = 4.8M PIDs/s
 - 80MHz / 4.8M PIDs = 16 cycles per PID



The figure above shows that TDM can be used on three levels in control loops of motion control systems:

- TDM within one controller (proportional-, integral-, derivative-, output-, feed forward-gain)
- TDM within one axis (position, velocity, current controller)
- TDM between different axes

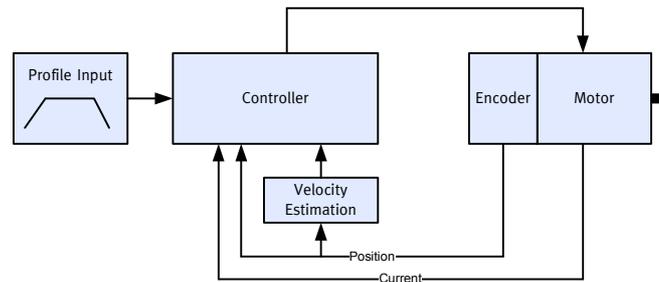
Because all calculations are done one after the other, BRAMs can be used efficiently to store parameters. This can lead to substantial savings in logic cells required for a design.

It can be shown that a TDM implementation allows to share many controller resources for a whole high-performance, 8-axis motion control system without exceeding reasonable clock frequencies for low-cost FPGAs.

The same applies for other parts of motion control systems. Trajectory integrators for example can also be implemented using TDM either explicitly in FPGA fabric or implicitly when running as SW on soft CPUs.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

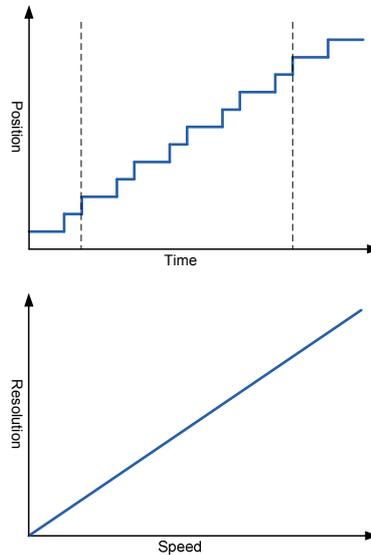
- The need for velocity estimation
 - Position can be measured
 - Velocity has to be estimated



Normally the use of special devices such as tachometers to measure the velocity is avoided because of the increased cost. Therefore, very often the velocity is estimated based on data available from position measurement.

The most common device for position measurement is an encoder, which has a limited position resolution. This leads to certain difficulties when estimating velocities.

- Conventional velocity estimation
 - Measure distance over a fixed time
 - Resolution +/- 1 increment
 - Resolution is proportional to speed
 - Scaling can not improve results at low speeds (excessive measurement time)



A simple way of estimating velocity is measuring the change of position during a known period of time (e.g. velocity controller period). This can be done very easily and does not introduce significant complexity to the system.

The velocity can be estimated based on two position samples, which leads to good results during constant velocity phases. But it may become problematic during fast acceleration phases.

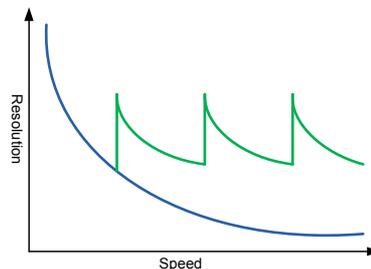
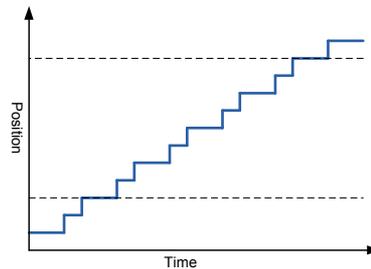
Another approach consists of measuring the velocity based on three position samples. This allows a correction for a constant acceleration with only a little additional effort.

Both estimations are based on the measurement of the position during a known period of time and have one common weakness: At slow speed their performance is poor. The reason for this is the limited position-resolution. If only 3 increments occur during the measurement interval, the measurement tolerance of +/-1 increment can lead to significant changes in the estimated velocity.

To improve this, the measurement time could be chosen very large but this is not reasonable because it would introduce significant delay in the velocity control loop. It is commonly known, that delay has a negative impact on controller performance or even affects stability.

- Velocity estimation using FPGAs
 - Measure time over a known distance
 - Resolution +/- 1 clock cycle
 - Resolution is reciprocally proportional to speed
 - Scaling at high speed is possible

$$v = \frac{s}{t}$$



Another approach to estimate velocity is measuring the time over a known distance. From the formula it is obvious that this will need much more computation effort since the non-constant factor (namely the time) is the divisor. Nevertheless it can make sense, in particular if the position measurement based approach does not lead to satisfactory results.

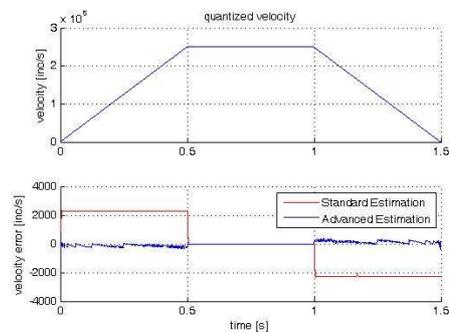
The advantage of measuring time is that the resolution in the time domain is very high (e.g. 20 ns at 50 MHz). If the distance is chosen properly, the error introduced by the limited time resolution is legible compared to the tolerances of position measurement devices.

The faster the drive moves, the shorter the time is and therefore the velocity resolution decreases. In contrast to the approach presented on the last slide, scaling is a reasonable solution. If the drive speed is doubled, the measurement distance is also doubled. The measurement time is about the same and therefore no additional delay is introduced.

The drawback of this approach is that there is no fixed sample rate in the velocity estimation because a new value is calculated only if a new encoder increment is detected.

Comparison

- 1 ms measurement time
50MHz clock
- Low speed (1k Inc/s encoder)
 - Position measurement: 1 bit
 - Time measurement: 16 bit
- High speed (5M Inc/s encoder)
 - Position measurement: 12-13 bit
 - Independent values every 1ms
 - Time measurement: 12-13 bit
 - Scaling 512x
 - Independent values every ~100us



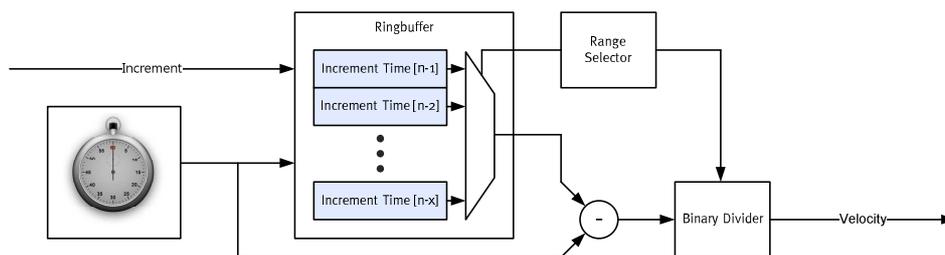
The slide shows a comparison between both approaches. For the approach based on distance measurement, a two-sample based variant is analyzed.

It shows that the approach based on time measurement is better at low speed and has about the same performance at high speed if scaled properly. The figure also shows that it behaves much better during acceleration phases.

Another point worth mentioning is that the time resolution is better at about the same velocity resolution. This can be used to speed up the controller and as a result of this to achieve better performance.

Implementation

- Perfectly suited for FPGAs
 - Ring buffer in block-RAMs
 - Too performance hungry for processors
 - Binary divider
 - Every increment has to be stored (up to 5M Inc/s)



The implementation of the position based velocity estimation is trivial: it only requires a differentiator (two point variant) or a three tap FIR filter (three point variant).

For the time based approach the implementation is slightly more challenging:

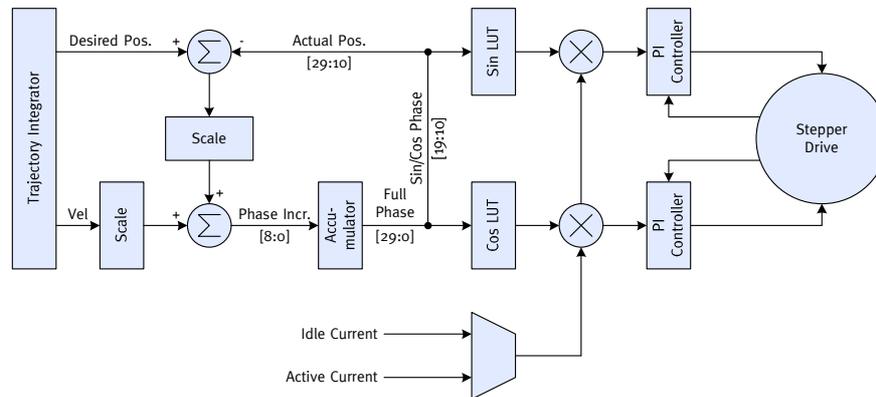
The timestamp for every increment has to be written into a ring buffer. Depending on the speed, a proper number of increments is selected as distance to base the estimation on. The time for this distance can then be calculated based on the current time and the time saved in the ring buffer. Afterwards the velocity is calculated with a binary divider.

FPGAs are perfectly suited for this implementation - they contain BRAMs for the ring buffer, multipliers for the scaling and the binary divider can easily be implemented in logic cells.

For CPUs this approach is hard to implement since the frequency of write operations to the ring buffer can become very high. However, there are other implementation concepts which can be realized with microcontrollers that contain specific peripherals. They base on the same principle but have some minor drawbacks.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- **DDS and Microstepping (Stepper Drives)**
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Setup to control a stepper drive



Controlling stepper drives is completely different from controlling DC or EC drives. The biggest difference is that they are, in terms of position control, not controlled in a closed loop but in an open loop. In some low-cost applications stepper motors are controlled in full-step mode which causes strong vibrations and noise. In this presentation we only consider the micro stepping mode that prevents vibrations and noise and its implementation using FPGA resources.

As widely known, controlling a bipolar stepper drive requires a sine and a cosine current, one for each coil. The stepper motor velocity is depending on the coil current frequency: the higher the frequency, the higher the velocity. To convert a velocity value to a frequency, an NCO can be used.

To achieve this, an accumulator counting with a variable step size is used. The step size is proportional to the velocity. Therefore the content of the accumulator is proportional to the position. Let's split up the accumulator into three parts for better understanding:

- Fullstep count (bits 29..18 in the figure)
- Microstep count within one fullstep (bits 17..10 in the figure)
- Sub microstep count within one microstep (bits 9..0 in the figure)

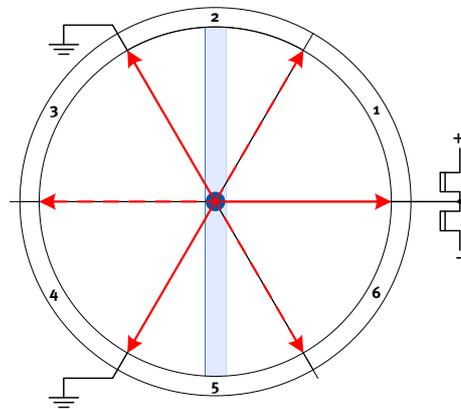
Because one sine period contains four full steps, the input to the sin/cos lookup table is built from the micro step count and two bits of the full step count. The other bits of the full step count are only needed to make the actual position available for high level control. The sub micro step count is necessary to allow the accumulator to run at the system clock rate and to make it possible to move very slowly.

The output of the sin/cos LUT is then multiplied with the desired peak current. Usually there are two different current levels, one for moving and one for holding a position. The current itself is then controlled the same way as for other motor types by a PI or PID controller.

To compensate for numerical issues or small position jumps due to trajectory integrator divergence during a move, the position deviation is calculated and added to the accumulator input. This prevents the trajectory integrator and the stepper controller from diverging.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- [Block Commutation and Field oriented Control \(EC Drives\)](#)
- Example Project
- Drive Control related Enclustra Products
- Conclusions

- Block commutation
 - Position is known +/- 30°
 - Efficiency @ 30° error = $\cos(30^\circ)=0.86$
 - Up to 14% of the energy is wasted
 - Torque is controlled by applying a positive or negative voltage to one coil



Some years ago, block commutation was state of the art in controlling EC drives. The advantage of block commutation is its simplicity, while ,as a drawback, it wastes up to 14% of the consumed energy. Today it is replaced more and more by the field oriented control (FOC) that has a higher efficiency.

Block commutation is discussed to give an introduction into EC drives and to show why it makes sense to use the FOC. FOC leads to the requirement for more processing power and thus puts FPGAs in favour for FOC implementations.

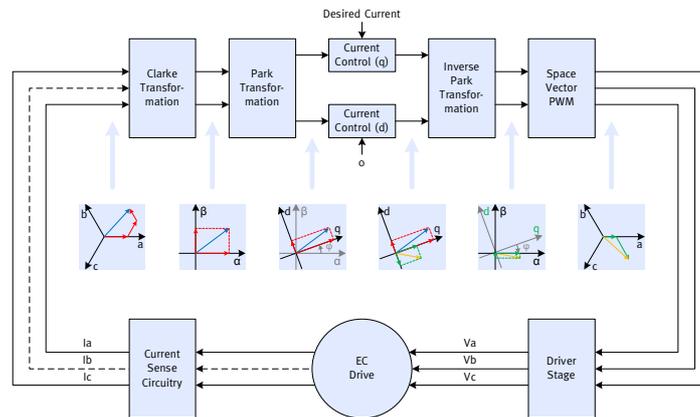
The position of the rotor within one rotation can be determined by reading the state of the hall sensors. Because there are only three hall sensors, the position is known with an uncertainty of +/- 30°. A full rotation is split up into six 60° sectors.

For each one of these sectors, two coils are grounded while a positive or negative voltage is applied to the third one to create a positive or negative torque (block commutation).

While this principle is very simple, it can lead to a bad efficiency of $\cos(30^\circ)=86\%$ if the rotor is not in the center of a sector. This can be problematic especially for high performance systems. For high power drives an even more problematic point is that a lot of power is wasted. In days of global warming and increasing energy efficiency people care a lot about the second point.

The drawbacks of the block commutation together with the availability of cheap processing power caused FOC to become state of the art.

- Field oriented control
 - Requires multipliers and sin/cos LUTs
 - Angle measurement based on the encoder



The basic difference between FOC and block commutation is that FOC measures the rotor angle based on encoder signals which have a much better resolution than hall sensors. FOC then applies a current to two coils instead of only one to produce a space phasor that is perpendicular to the rotor and therefore leads to a theoretical efficiency of 100%.

To achieve this, knowledge of the actual space phasor is required. Therefore two phase currents are measured. The third one can be calculated from them ($I_1 + I_2 + I_3 = 0$) and therefore only contains redundant information. Note that it is important to execute both measurements simultaneously to get consistent data.

In the Clarke transformation, the space phasor is mapped to a Cartesian coordinate system by simple trigonometry. In the Park transformation the currents with respect to the rotor angle are calculated. The flux producing current is called I_d , the torque producing current is called I_q .

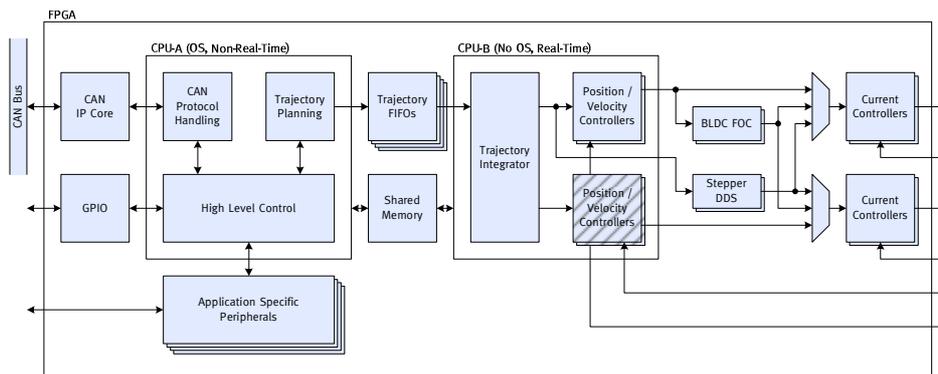
I_d does not produce any torque and should thus be zero. The torque producing current has to be set to the value requested by the velocity and position controllers. Both parts of the current are controlled by a standard PI controller.

The outputs of the current controllers are then transformed back to the Cartesian coordinate system (inverse Park transformation). The conversion from the Cartesian coordinate system to a duty cycle for each phase is done directly in a block called SVPWM that controls the power stage.

The whole FOC has to run at the same speed as the current controllers which are the fastest part of drive control systems. Therefore FOC needs a significant amount of processing power. The operations that are required for the transformations (multiplications, additions and sin/cos lookup tables) are very FPGA friendly. This leads to the conclusion that FOC can be easily implemented in FPGAs (and DSPs for not too high control rates) while it is not very CPU friendly.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- [Example Project](#)
- Drive Control related Enclustra Products
- Conclusions

- Drive Control Plattform
 - 4-DC drives or 2 EC drives or 2 stepper drives or any combination thereof
 - Current controller at 100 kHz, position controller at 5 kHz
 - Many application specific peripherals



Many of the parts considered in this presentation are also present in customer projects realized by Enclustra GmbH.

This slide gives an overview of one specific project to show a real-world implementation of the theory covered in this presentation. The aim of this project was to realize a customer specific motion control platform with CAN interface which is able to control multiple axes with different types of drives.

A two CPU approach was chosen because of several reasons:

- Much CPU power required for high level control
- Encapsulate real-time and non-real-time software
 - CPU-A is programmed by the customer
 - CPU-B is programmed by Enclustra
- Allows using an OS for high level control without having any overhead in the real-time part

This system also contains a lot of application specific peripherals such as step loss supervision, drive power dissipation integrator, I/O crossbar, position interrupt generator, drive position latching on several events, etc.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- [Drive Control related Enclustra Products](#)
- Conclusions

- Mars MX1/MX2 FPGA Module
 - Low-cost, low-power Spartan-6 FPGA
 - SO-DIMM form factor
 - Single supply voltage
 - 16 MB quad SPI Flash
 - 256 MB DDR2 SDRAM
 - Dual Fast Ethernet (MX1 only)
 - Real time clock (MX1 only)
 - Gigabit Ethernet (MX2 only)
 - Dual multi-gigabit transceivers (MX2 only)
 - PCIe endpoint (MX2 only)



The Mars MX1 module in particular is a perfect platform for implementing drive control systems. Beside its optimization for low cost and SOPC systems, it also contains two ethernet physicals to support real time Ethernet standards such as EtherCAT.

- IP
 - Velocity Estimator IP
 - Delivered first evaluation netlist
 - Drive IP
 - Under development
- Drive FMC
 - FMC LPC connector
 - 6 Halfbridges
 - Up to 3 DC-, 2 EC- or 1 stepper drives
 - Low cost single channel ADC
 - 12 channel ADC for multi axis systems
 - Up to 36V I/O

The Velocity Estimator IP does implement the velocity estimation based on time measurement described earlier in this presentation.

The aim of the drive IP core is to implement a full motion control system including all controllers, FOC and stepper motor control. Note that this IP core is still under development (May 2011).

Because of its flexibility, the Drive FMC daughtercard allows rapid prototyping for many different drive applications including the control of different motor types. It also contains two different types of current sensing: One for absolute low-cost single-axis systems and one for cost optimized multi-axis systems.

- Overview
- Task Partitioning
- Trajectory Planning
- Trajectory Integration
- Control Loops in FPGAs
- Velocity Estimation (DC/EC Drives)
- DDS and Microstepping (Stepper Drives)
- Block Commutation and Field oriented Control (EC Drives)
- Example Project
- Drive Control related Enclustra Products
- [Conclusions](#)

- FPGAs allow:
 - Highly integrated ...
 - ... high performance drive control ...
 - ... at relatively low cost
- For low end applications DSPs and drive control optimized CPUs may also be reasonable
- Next generation FPGAs will contain some important features
 - Hardwired CPUs (Zynq)
 - CAN interface & dual Ethernet (Zynq)
 - Integrated dual ADCs (Zynq, 7 Series)
 - Lower cost (General)

FPGAs are becoming more and more the technology of choice to realize drive control systems. DSPs and drive control optimized CPUs may still be the better solution to implement lowest cost or low end drive control systems but with every new FPGA generation, this domain gets smaller.

The actual next generation FPGAs are a very big step into this direction: They contain integrated dual ADCs which fit perfectly to the requirements of drive control systems. This will cut down BOM part count and costs while the FPGAs itself also become cheaper. The new Xilinx extensible processing platform (Zynq) even integrates some more features that are widely used in drive control applications: Two hardwired CPUs as well as CAN and Ethernet interfaces for field bus access. This is a big step towards programmable single chip solutions for drive control.



Oliver Bründler
Enclustra GmbH
bruendler@enclustra.com
Fon +41 43 343 39 40

Slides in PDF format:
<http://www.enclustra.com/de/company/publications/>

Dear Attendant,

I hope I could achieve the aim of this presentation and give you a brief overview over state of the art drive control with a special regard to the FPGA technology.

If you have any questions about general drive control or regarding your own projects, do not hesitate to contact me.

Thank you for your attention,

Oliver Bründler