Motor Control - Stepper Motor Driver for Smart Gauges

AN2197

Author: Victor Kremin
Associated Project: Yes
Associated Part Family: CY8C24xxxA, CY8C27xxx
GET FREE SAMPLES HERE
Software Version: PSoC Designer™ 4.2
Associated Application Notes: AN2161

Application Note Abstract

This Application Note demonstrates using the PSoC® mixed-signal array to drive a low-power stepper motor for smart pointer gauges.

Introduction

The world is digital nowadays, and most information is represented in numbers. However, human nature is more "analog" and better represented the old-fashion way, using pointer gauges and bar graphs.

Pointer gauges can be found in various industrial applications. Automobiles, trains, and even modern aircraft dashboards emulate analog gauge functionality on control flat panel plasma or TFT screens. It doesn't look like good ol' pointer gauges will disappear in the near future.

Various techniques can be used to control a pointer gauge. The most popular technique is to use a mechanical system, which consists of a turning coil mounted outside a two-pole permanent magnet. The applied DC current causes a magnetic force that rotates the coil and associated gauge pointer. Springs limit the coil rotation angle and the stable pointer rotation angle is in direct proportion to the coil current. Such a gauge can be equipped with an oil damper to suppress oscillations during coil angle setup and improve the system's mechanical stability with respect to vibration. This method has limitations in the operational temperature range because oil viscosity changes with temperature, causing the gauge to be unstable amid vibrations.

Other gauges use a bimetallic plate with a heater. This type consumes much current during operation and readings are dependent on environmental temperature.

An alternative approach uses two quadrature-located coils to set the pointer position. In this system, the pointer rotation angle is determined in relation to the coil. A mechanical damper is still required to prevent pointer flicker due to mechanical vibrations at setup time.

A perfect way to control a pointer gauge is to use a stepper motor. Conventional stepper motors have large rotation steps and non-uniform torque distribution within steps that require using expensive gears and produce audible noise signals during operation. The stepper motor-based gauge driver is equipped with a pointer position sensor to check initial pointer position, but additional sensors increase the driver costs.

Motor Driving Principles

Today, many companies provide stepper motors for gauges. These motors are characterized by small size and low power and can be driven directly by the microcontroller or by level translators. Most motors have built-in gearboxes, which increase motor torque. Such motors are SONCEBOZ MM39 (6405E-1550) and NMB part #PM20T. These motors are controlled by the quadrature sin/cos analog signals to provide smooth rotor rotation.

When a two-pole permanent magnet is used in the motor rotor, the rotor mechanical step is 90° when single-phase electric pulses are applied to the motor windings. Therefore, the microstep technique is necessary to control this motor in gauge applications.

This can be achieved by applying the cos/sin analog current signals to the coils. Because $\forall \phi, \cos^2 \phi + \sin^2 \phi = 1$, and motor rotor flux linkage is the same for any rotor angle by the mechanical construction of the motor, torque is constant. To use a gauge for analog values measured digitally, it is necessary to divide each motor mechanical 90° step by a predefined number of microsteps. In this design, each mechanical 90° step is divided by 32 microsteps.

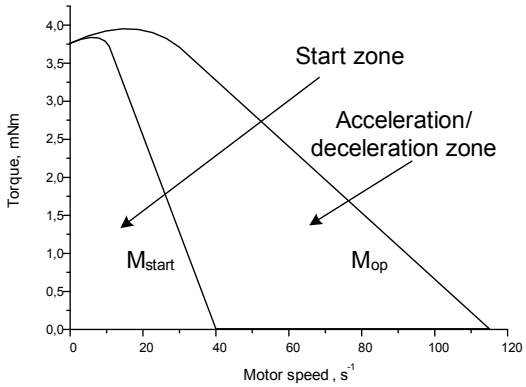
When a motor is used in the gauge application, the principles of sensor or sensor-less synchronization can be used to detect pointer stops. Motor manufacturers recommend driving the motor in the synchronization phase with rectangle pulses (full-step mode) and reading the back-EMF from the windings. When the rotor turns, a back-EMF signal is produced, when a stop is reached, there is no inducted voltage. The design in this Application Note uses this principle to detect pointer stop.

Driving motors is simple. In this Application Note, two analog quadrature sin/cos signals are generated using the double pulse-width modulator (PWM) or a digital-to-analog converter (DAC). The driving profiles should be determined as well. Motor speed can be a constant or variable with respect to acceleration and deceleration phases.

For stepper motors, the startup frequency is much less than maximum operation frequency. Therefore, the constant speed profile does not provide the minimum pointer setup time. The variable speed driving profile does not have this limitation and allows full use of motor possibilities and use of the minimum pointer setup time.

There is a limit to the maximum rotation acceleration as well. This limit determines motor acceleration and deceleration times; the limit comes from the limit in the magnetic force value. Note, if stepper motor control frequency (both startup and operation) exceeds predefined limits, the motor can skip steps and pointer position may lose synchronization with the control sequence. Therefore, the maximum acceleration and startup times as well as operational rotational speeds must not be exceeded.

Figure 1. Stepper Motor Torque vs. Motor Speed



In stepper motor gauge design, it is possible to select different motor acceleration and deceleration schemes. One possible solution uses a digital low-pass filter (LPF) to gradually increase the rotational speed during the motor acceleration phase and to apply the same filter during the deceleration phase. When this scheme is used, the absolute value of the acceleration must be within the allowed bounds of the motor.

Another scheme can use a constant acceleration profile. This type of profile is useful when the rotation speed is increased at constant acceleration during the acceleration phase, with corresponding deceleration during the deceleration phase. Rotational speed is constant when the speed reaches a predefined threshold. The proposed motor driver uses a constant acceleration drive profile. This profile is sometimes called a trapezoidal profile, since it is in the shape of a trapezoid.

This driver can be implemented with a microcontroller and some application requirements. The following components are necessary:

- A double-PWM or DAC to create the phase coils' quadrature signals;
- A variable frequency generator to generate the phase current values' updating events to determine the rotational speed;
- A speed control system to operate rotational acceleration and deceleration.

Modern microcontrollers have PWMs. The programmable interval timer can be used as a variable frequency generator. However, change in linear motor rotation speed corresponds to a hyperbolic timer period curve, which requires a high-resolution timer.

PSoC provides an excellent solution with its flexible internal analog user modules. The voltage-to-frequency converter (V/F), together with a DAC, creates a programmable, variable frequency signal generator with a constant frequency step. This method is used in this Application Note.

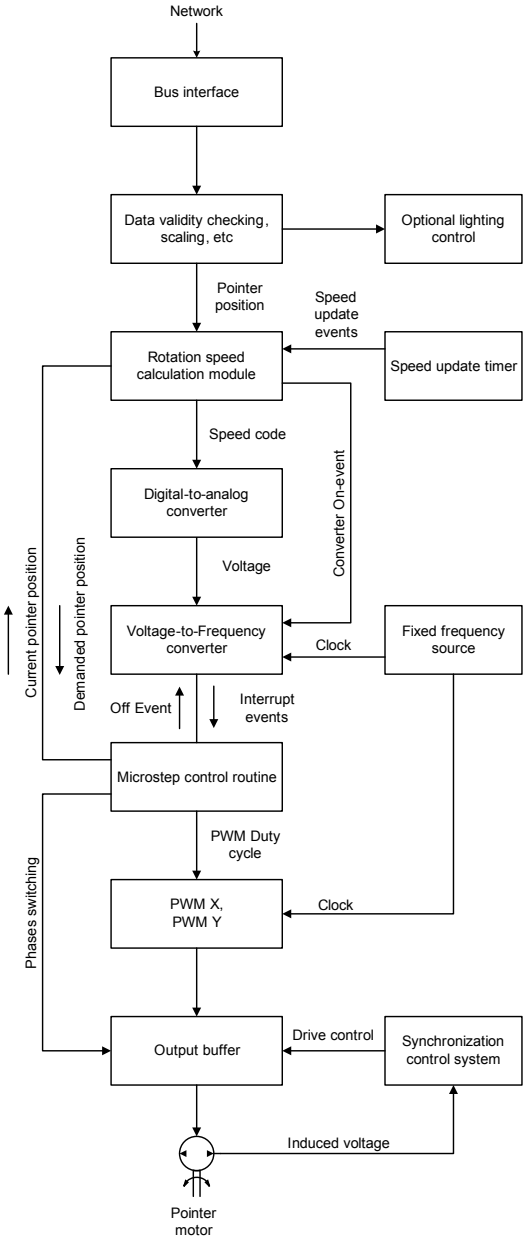
Most modern applications require networked gauges, where all gauges are connected to a common bus with minimal wires (the modern vehicle contains many buses inside, such as CAN, LIN, J1587, and others). In such cases, the bus interface reads and interprets the bus data, and selects the commands to be processed by a particular gauge.

A standard UART interface has been used to control the gauge driver in this example. The end user can use end application-specific protocol.

Figure 2 shows one possible microcontroller-based driver implementation.

Note The conventional Unified Modeling Language (UML) notation was used to mark the relations between blocks.

Figure 2. Driver Cooperation Diagram



The driver can be connected to a dedicated bus. The network interface receives the data from the bus, decodes it, checks the incoming packets' integrity, and separates suitable data for a particular gauge in the network. The received data is parsed, validated, and scaled to the pointer microsteps or other suitable processing value used to control pointer movement. Note that in many vehicle applications, the gauge is equipped with the several illumination LEDs and one or more status LEDs (low fuel, overheating, alarm, etc.). These LEDs can be controlled via the bus as well.

The speed calculation module analyzes the current and required pointer positions to generate the actual motor rotation speed; the speed adjustment is periodically initiated using a dedicated interval timer. The motor rotational speed value is calculated using Equation (1) for each timer update:

$$v_i = v_{i-1} + a_i,\ a_i = \{a_0, 0, -a_0\} \qquad \text{Equation 1}$$

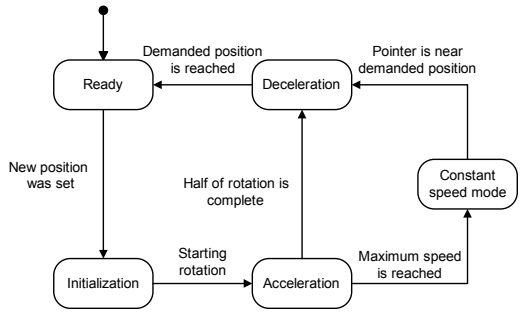
v_i and v_{i-1} are speed values for i and $i-1$ iterations, respectively, and a_i is the acceleration value, which can accept only three possible values:

- Fixed positive during acceleration stage.
- Zero during constant rotational speed stage.
- Fixed negative during deceleration stage.

Upon motor startup, positive acceleration is selected and half of pointer movement distance is used to mark the switching on of deceleration stage. When the pointer speed reaches the predefined threshold, acceleration drops to zero and the constant rotational speed stage starts. At this time, the current pointer displacement overrides the previously calculated value for determining the pointer deceleration start position. The deceleration stage begins when the distance-to-destination position is less than the previously calculated value; the acceleration is set negative for this stage. The proposed algorithm provides symmetric acceleration and deceleration profiles for both small and large pointer displacement, regardless of rotation speed and the maximum-allowed value.

Figure 3 shows the speed control module diagram. *Ready* is the default stage. When a new position command is received, the driver enters the *Initialization* stage, where the internal control variables are initialized. Next, is the *Acceleration* stage, during which the motor rotation speed increases linearly. When rotation speed reaches the predefined maximum value, the driver enters the *Constant Speed Mode* stage. If the pointer is close to the set position, the driver enters the *Deceleration* stage, during which speed decreases linearly. Note that the driver enters *Deceleration* stage directly from *Acceleration* stage when the pointer completes half of the required rotation angle and the speed is less than the predefined maximum value. This occurs frequently at small pointer displacements.

Figure 3. Speed Control Module State



The V/F is used to generate variable frequency interrupts to call the microstep control routines. The input voltage is calculated to provide the interrupts' frequency proportional to that from Equation (1). The speed value is calculated suing Equation (2).

$$U_i = \frac{v_i}{K_s} \qquad \text{Equation 2}$$

K_s is the scale coefficient.

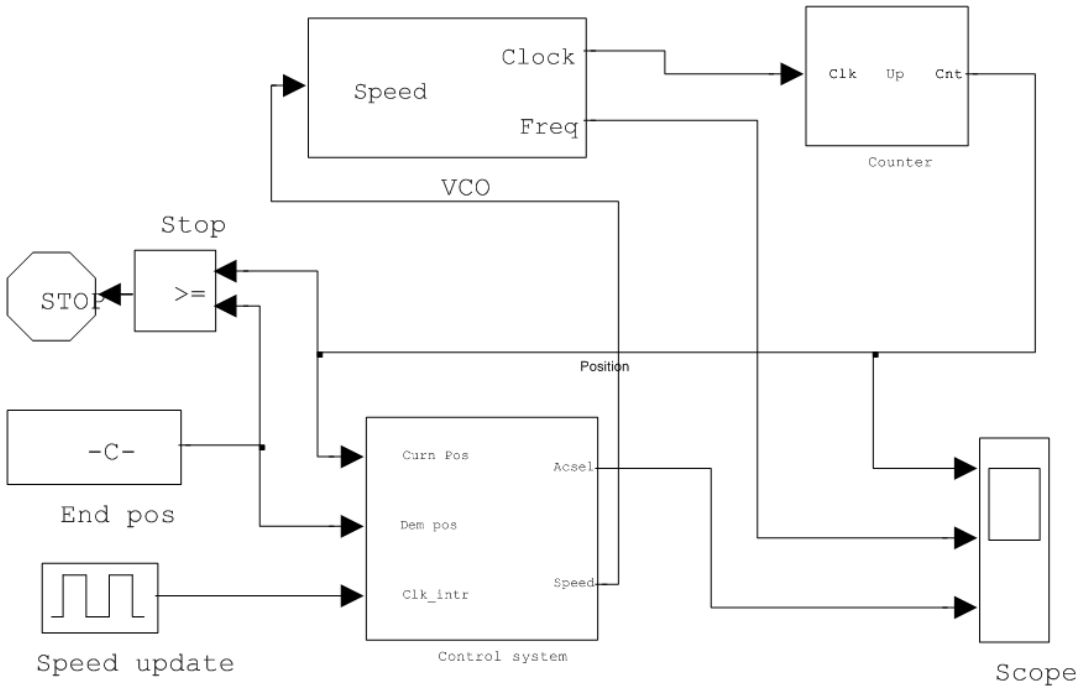
The microstep control routine does several things: it adjusts the motor PWM duty cycle values, it switches the direction of the motor windings when needed, and it compares the current pointer position in microstep units to that of the required position. When these values are equal, the V/F stops, PWM units are turned off (or PWM duty cycle values can be proportionally reduced to avoid false rotations in strong gauge vibrations), and pointer movement is considered complete.

The synchronization control system performs initial pointer synchronization by analyzing the inducted voltage to detect the stop point.

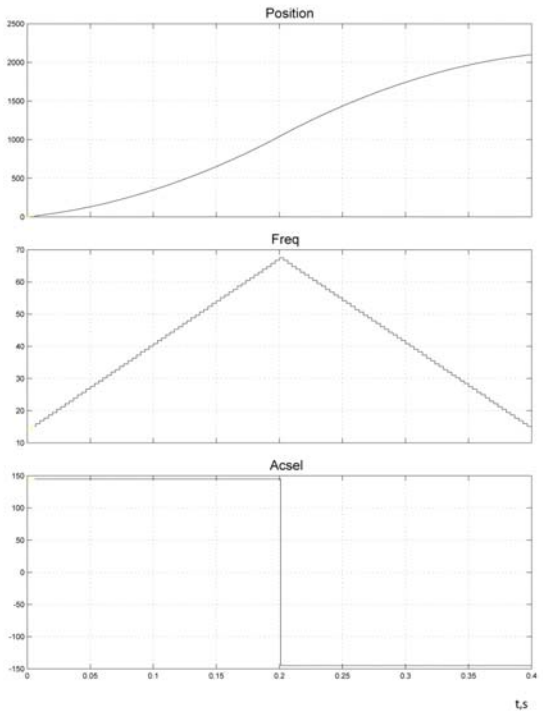
The motor control system was simulated in Simulink to study the quantization effects and optimize parameters in the control algorithms. Figure 4 illustrates a top-level model control system and simulation results for various pointer displacements. The simulation results show the accuracy of the control algorithms and match driver experiment test results.

Figure 4. Motor Driver Simulation

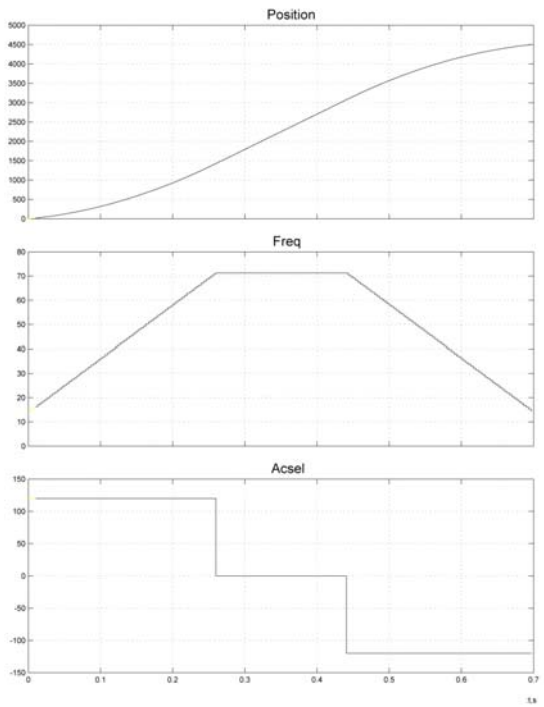
(a) Driver Simulink Top-Level Model, (b)-(c) Simulation Results for Various Pointer Displacement



(a)



(b)



(c)

Driver Schematic

Figure 5. Driver Schematic

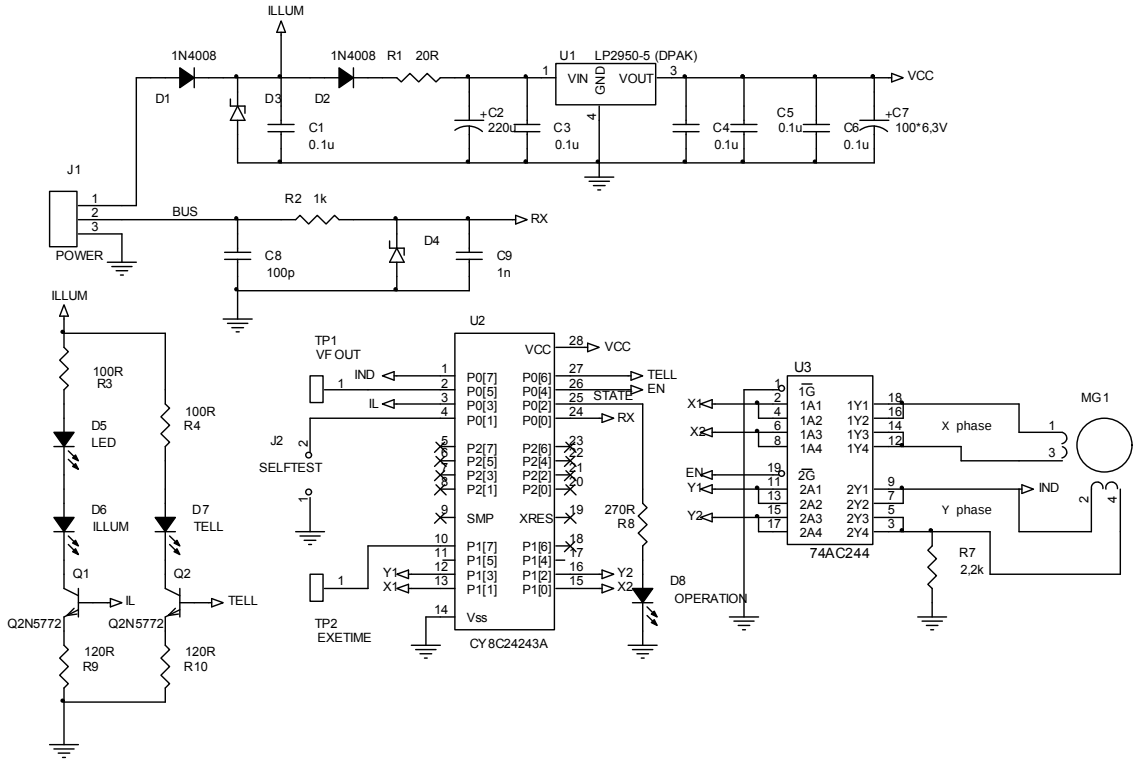


Figure 5 shows the device schematic.

This driver was designed for vehicle applications with supply voltages between 8V and 18V and a single-wire control bus. The 28-pin PSoC CY8C24423A is used in the prototype. End applications can use a 20-pin CY8C24223A since the port 2 pins are not used.

The driver consists of a power regulator (U1), a bus input protection circuit (R2-D4-C9), a PSoC (U2), a motor driver (U3), and two voltage-to-current converters (Q1 and Q2). The bus driver (U3) is used to drive the low-power stepper motor and increase the load capability of the two outputs connected in parallel for each motor phase. The U3 internal diodes protect the driver from inductive spikes. Note that the PSoC device has an asymmetric load capability (12 mA for source current and 25 mA for sink), which requires using an additional driver. The low-cost bus translator (U3) does this perfectly. One half of the translator can be disabled by using an EN signal during the synchronization process when the inducted voltage is read from the Y phase.

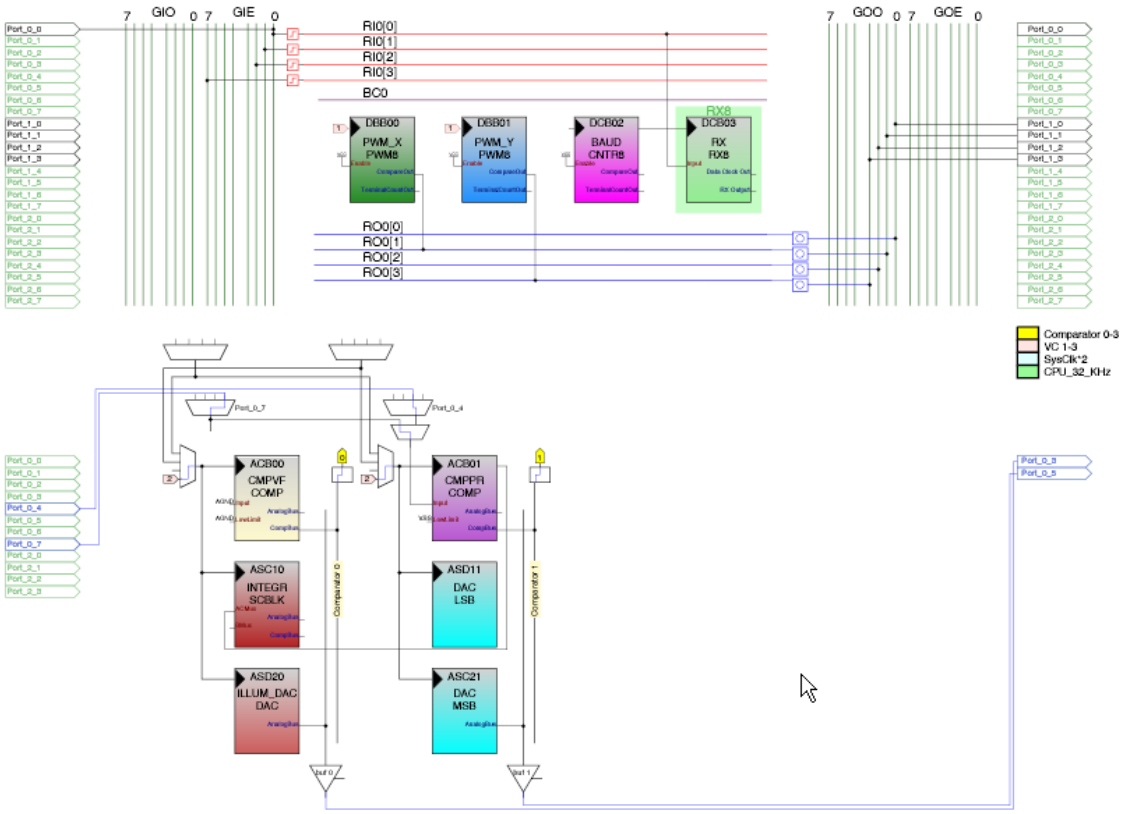
R7 pulls down the line for correct Y-voltage reading when the Y-phase outputs are disabled.

Two current sources are used to drive the illumination (D5 and D6) and status (D7) LEDs. The illumination current is controlled by a PSoC internal DAC, which allows gradual tuning of LED brightness with respect to external commands. The status LED is controlled by logic. The current sources provide constant brightness in case of power supply or temperature variations. These LEDs are directly connected to a battery line to reduce the linear regulator (U1) power dissipation. D8 is used internally; it starts blinking when the gauge is not addressed on the bus within 1 or 2 seconds. Constant LED lighting indicates the motor synchronization process. When the bus master selects the gauge, this LED is off.

There are two test-points on the schematic, TP1 and TP2. TP1 is the rotation speed of the DAC output. TP2 measures the execution time of code fragment using dedicated macros.

PSoC Internals

Figure 6. PSoC User Module Placement



The PSoC internal structure is shown in Figure 6.

The X- and Y-phase PWMs are placed in blocks DBB00 and DBB01, respectively, and their signals are connected to the phase outputs via LUTs. The LUT functions are charged at runtime to properly route the phase signal to the corresponding pins during motor operation. The PWM frequency is approximately 19 kHz, which assures acoustic noise-free operation. It is possible to adjust the required PWM frequency by changing the VC1 and VC2 divider values with respect to the motor manufacturer's recommendations.

The V/F is placed in blocks ASB00-ASC10. The converter output generates periodic interrupts using the comparator bus interrupts. The converter operation is described in detail in Application Note *AN2161 Voltage-to-Frequency Converter*.

The 9-bit DAC is used as the V/F signal source. Because the DAC internal output alternates between AGND and the set level, each column clock cycle and V/F sample the input signal during both switching capacitor phases. The DAC output is passed to the analog bus and then to the AMUX input of the V/F via the PGA. The PGA is a programmable threshold comparator, to which the PSoC changes using dynamic re-configuration. Writing directly to the control registers performs the re-configuration.

The programmable threshold comparator is used to control motor induction voltage during the synchronization process. The comparator is queried in software during synchronization. It is placed in block ACB01. When unused, the comparator block is configured as the PGA.

This demonstration uses a UART-controlled exchange protocol, where all messages are encoded in text strings. HyperTerminal can be used to send commands to the driver. The user can use any other appropriate interface (SPI, I2C, etc.) to control the driver. For example, in vehicle dashboard applications, one master can control several slave gauges via an I2C bus, where the I2C master uses LIN bus slaves.

The DAC that controls illumination brightness is placed in ADS11. The 6-bit DAC is used to set the LED brightness level.

The VC3 interrupts generate the periodic speed-update intervals. In this design, the interrupt frequency is 4800 Hz. The speed update event is triggered every 16 interrupts, so the rotation speed value is recalculated once every 3.3 ms. The VC3 interrupts are also used to form the phase excitation pulse duration during motor synchronization. VC3 uses a stable, divided high-speed generator signal (accuracy is ± 2.5%). This assures that the acceleration value is set accurately for proper motor operation.

Sleep timer interrupts are used to form the blinking D8 events and update the bus exchange timeout. The fact that the sleep timer is driven from the low-accuracy (± 50%) internal low-speed oscillator is not important for these non-critical operations.

Note that the CPU clock is 12 MHz, such that at a 24 MHz clock, the maximum junction temperature is 82°C, with an ambient temperature of only 70°C. PSoC allows a maximum ambient temperature of 85°C for a clock rate of 12 MHz or less.

The Software

The driver firmware consists of several elements:

- Bus Interface
- Command Parser
- Time Management
- Pointer Positioning Control
- Pointer Synchronization
- Self-Test Function

The bus interface decodes and interprets the messages from the bus. In this demonstration, each device is characterized by its own address and can parse the following commands:

- Turn on/off the status LED.
- Turn on/off illumination LEDs and gradually set brightness level (62 different levels and an off state are supported).
- Stop pointer synchronization.
- Set pointer position in microsteps.
- Set gauge parameter value in chosen internal unit (km/hour, Celsius, etc.).

The pointer is synchronized to the initial position (internal stop) at gauge power up. However, the synchronization process can also be initiated by sending the appropriate commands to the gauge.

When a set parameter command is received, the parameter value is checked for upper and lower bounds and linearly scaled to be in the chosen microstep unit. In this demonstration, the allowed pointer displacement is limited to 4400 microsteps; this value is calculated from the motor's mechanical construction, especially the gearbox reduction ratio. The calculated microstep value is checked again to eliminate any errors in the scale coefficient settings. Equation (3) shows the calculation.

$$M_v = \min\left[M_{\max}, \max\left(M_c, M_{\min}\right)\right];$$
$$P_{ms} = \min\left[P_{\max}, \max\left(\frac{2^{K_p}}{K_s} M_v + M_{off}, 0\right)\right] \quad \text{Equation 3}$$

M_c, M_{min}, and M_{max} are the received, minimum, and maximum allowed parameter values, respectively. P_{ms} and P_{max} are the calculated and maximum permitted pointer positions in microsteps. K_p, K_s, and K_{off} are scale coefficients. The user can adjust the parameter conversion formulas or use other data types to serve special parameter representations.

In the current UART-based protocol, each message is a carriage return terminated string that consists of three parameters. All parameters are separated by spaces: "***Node_Address Command Command_value***."

All integers are in hexadecimal format.

- Address field: number from 0..FFh.
- Command field: one letter. All supported commands are given in Table 1.
- Value: two-byte unsigned integer value.

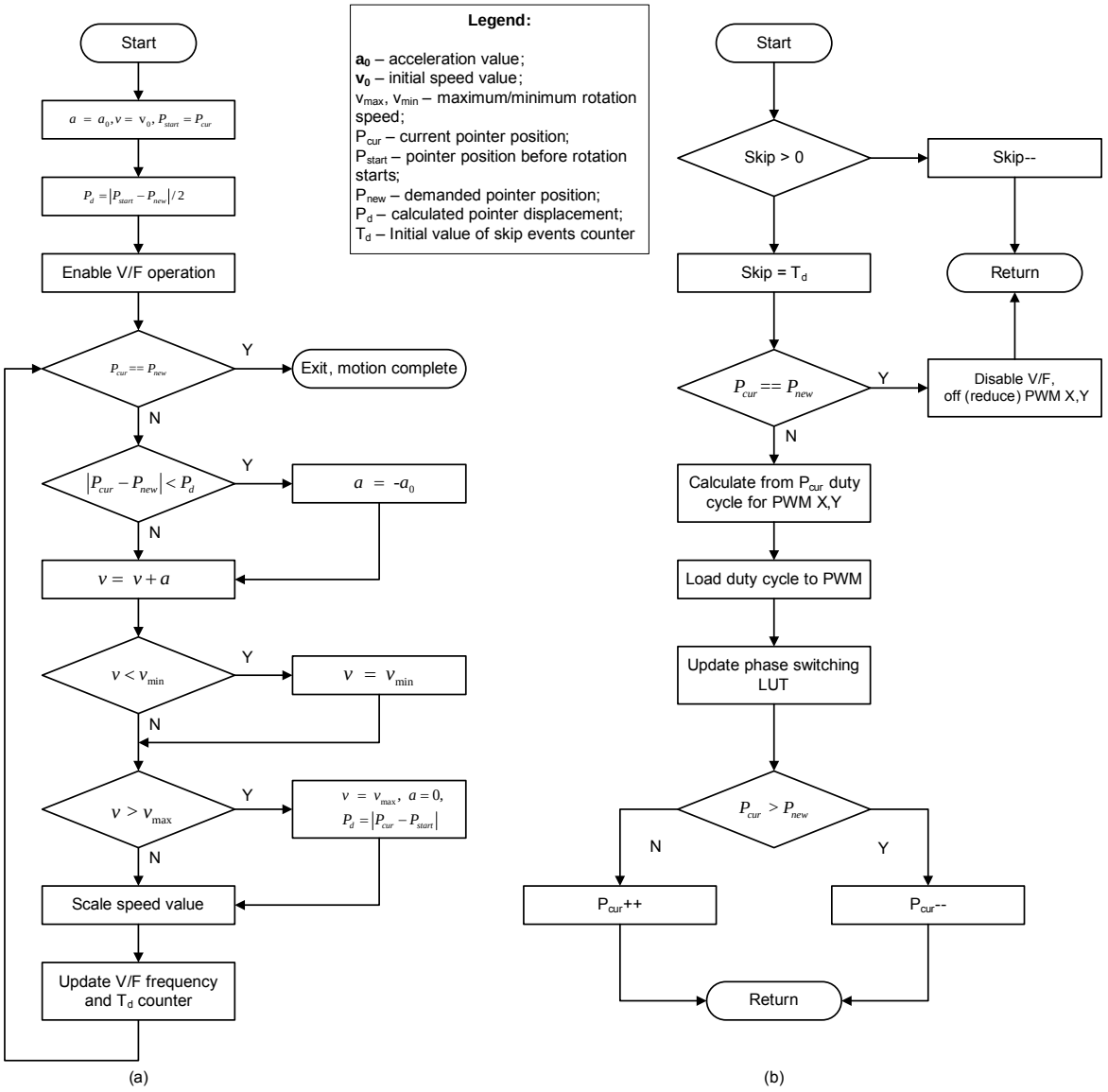
Table 1. Command Descriptions

Command Letter	Command Description
'P'	Set parameter. Allowed values are 0-350.
'I'	Initialize motor. Set pointer to stop using back-EMF synchronization. Value is not important.
'M'	Set pointer position in microsteps. Valid values are 0-4400.
'A'	Set acceleration ratio. Valid values are 0-150.
'B'	Set illumination LED brightness. Valid values are 0-62.
'L'	Turn on/off the status LED. Non-zero value turns on the LED. Zero turns it off.

A command example is: "10 P 12C," meaning that the node address is set to 10h, the command is "set parameter," and the parameter value is 12Ch.

The motor positioning control firmware consists of two main routines: speed update and microstep control routines, which are shown in Figure 7.

Figure 7. (a) Speed Adjustment and (b) Microstep Control ISR



The Figure 7(a) shows the speed control algorithm, which is described in the **Motor Driving Principles** section on page 1. Figure 7(b) shows the flow of the microstep timer ISR. The minimum V/F output frequency is approximately 128*1.5 Hz (limited by the offset voltage and DAC output voltage swing limits). This can be extended for very low frequencies using an additional software counter. This allows very low pointer positioning speeds, which can be useful for special applications. The current implementation allows a minimum pointer rotational speed of less the 1 RPM. The number of interrupt events for skipping is calculated in the speed control routine.

To calculate the sin/cos duty cycle values for the PWM sources, the 5 least significant bits are separated from the current pointer position variable and used as indices for the quarter period sin LUT. Thus, only 32+1 bytes are allocated for storage of this table in PSoC's Flash.

The PSoC LUTs were used as the PWM signal multiplexers to allow bridge coil control and maximize the usage of the power supply voltage. Figure 8 illustrates this.

Figure 8. Motor Phases Control

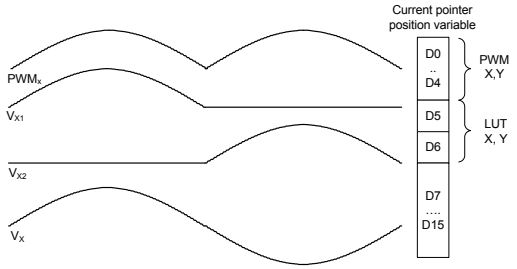
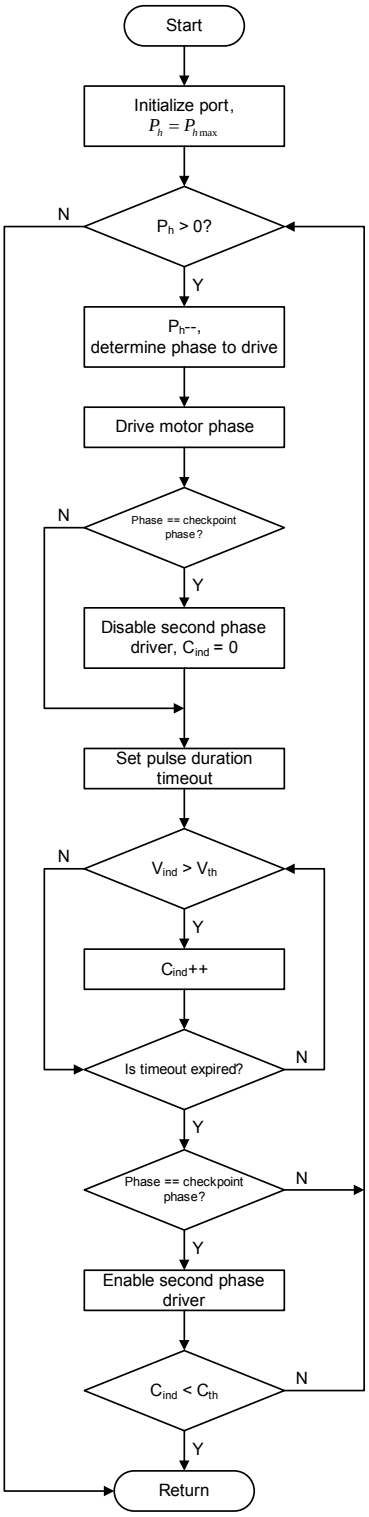


Figure 9 shows the synchronization algorithm, which provides sensor-less pointer rotation to the internal stop.

Figure 9. Pointer Synchronization Mechanism



Stepper motors for gauges can have special mechanical construction to ensure that when pointers stop, the rotor magnet poles are located close to same phase coils for any motor in a series. This simplifies the synchronization logic by only reading the inducted voltage during one phase interval (rotor step).

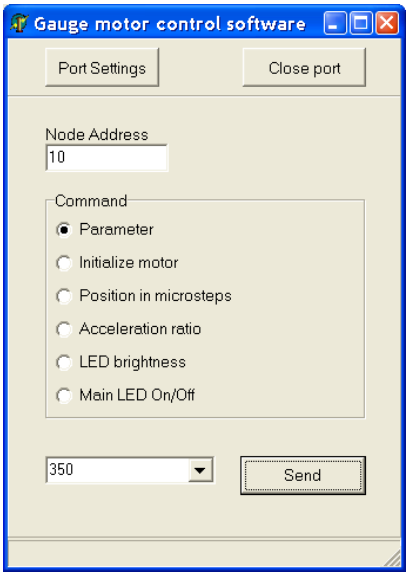
The motor in this design operates in full-step mode (wave drive mode) during the synchronization process and the inducted voltage is read only when phase x1 is driven (see Figure 5). A simple digital filter is used to suppress the noise caused by inter-coil capacitance. The motor rotor is considered rotating when the voltage on the sense coil is greater than the predefined threshold for more than C_{th} samples during the checkpoint phase excitation time.

The self-test feature allows users to test the gauge without active bus commands. Jumper J2 should be shorted in this case. After self-synchronization stops, the pointer is commanded to reach various positions in microsteps, and various illumination LED levels are set in series. A range of acceleration values are set to simulate different damping ratios as well. The status LED flashes anytime the pointer reaches the demanded position.

PC Test Software

To simulate a bus interface, simple test software was written using Borland Delphi 7 and runs with Microsoft Windows. The software remembers the previously entered numerical values in the drop-list box, which allows for easy repetition of previously entered commands.

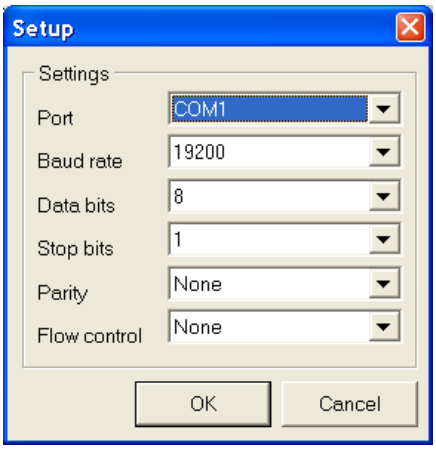
Figure 10. Gauge Control Test Software



Note The Variable field (next to the **Send** button) can be entered in decimal or hex format (using the x prefix).

The software interacts with the device using a PC serial port. Valid settings are shown in Figure 11. Note that the RS232-CMOS Level Translator is **not** required to test the gauge when using the PC. By connecting the TX line with the bus pin of Connector J1 (see Figure 5), the internal circuit provides all of the required protection.

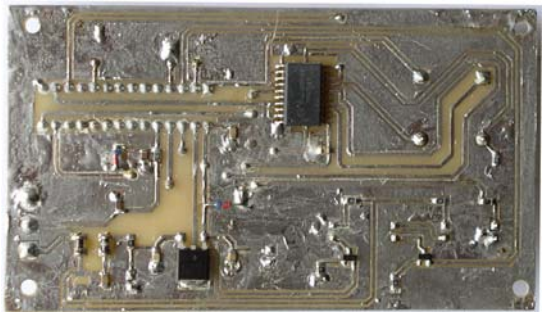
Figure 11. GUI COM Port Settings



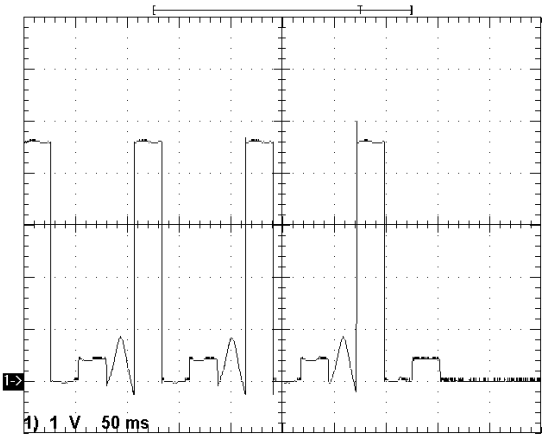
Possible Design Modifications

The proposed driver can be adapted for other demands such as servo control and various industrial applications. The synchronization technique in this Application Note can be adapted to conventional stepper motors with non-athwart coil placement. The digital filter can remove the parasitic spikes when the drive phase is excited.

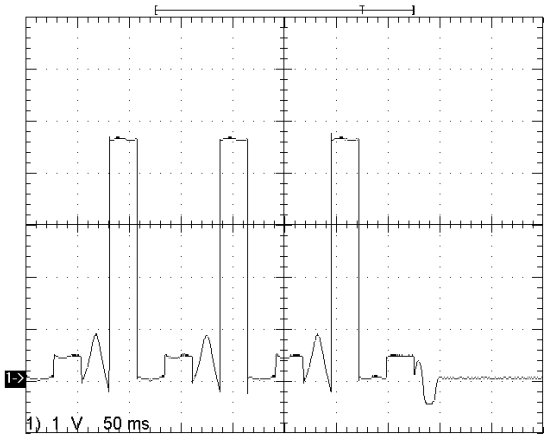
Appendix A. Driver Prototype Photograph



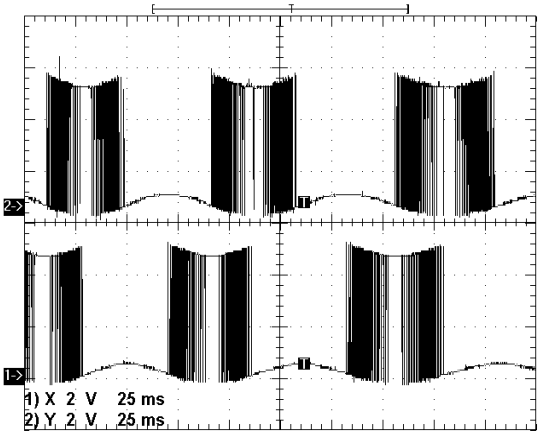
Appendix B. Scope Images



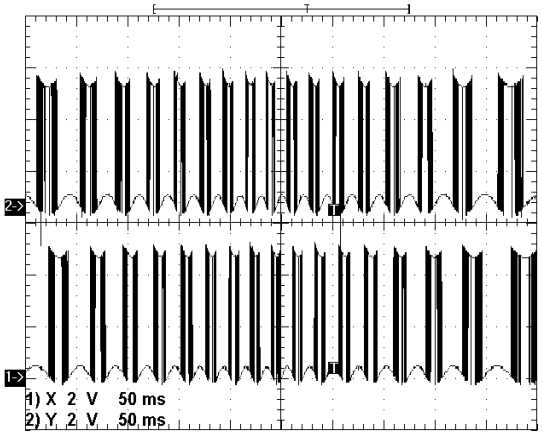
(a)



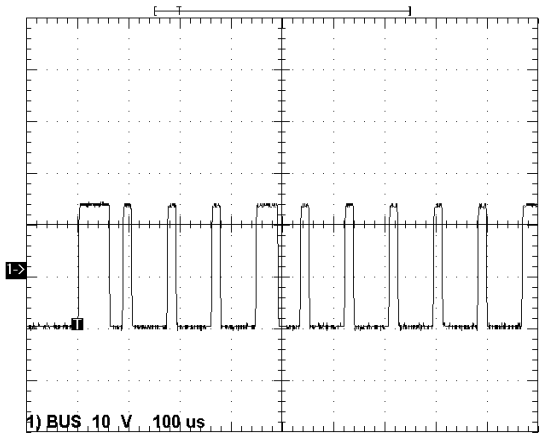
(b)



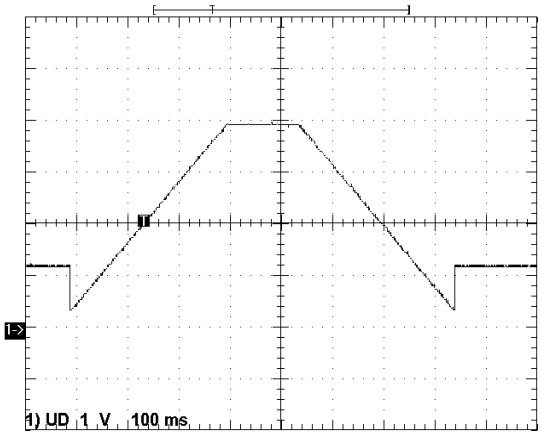
(c)



(d)



(e)



(f)

Image Remarks:

- Images (a) and (b) show the motor synchronization process for different pointer inertia moments. The stop moment is clearly displayed.
- Images (c) and (d) show voltage on the motor pins for two different phases. The increasing lower bound and decreasing upper bound correspond to the voltage drop on the opened MOSFET due to coil current increase according to the sine law.
- Image (d) shows the rotation speed acceleration/deceleration.
- Image (e) shows the RS232 communication signal.
- Image (f) shows speed control DAC output voltage.

Note Rotation starts and finishes at some intermediate level when the DAC output voltage is set above some minimum voltage to get a very small internal control step frequency by value control using the software skip counter.

References

1. "Handbook of Small Electric Motors," William H. Yeadon, Alan W. Yeadon, McGraw-Hill, 2001

About the Author

Name: Victor Kremin
Title: Associate Professor
Background: Victor earned his Radio Physics diploma in 1996 from Ivan Franko National Lviv University, PhD degree in Computer Aided Design systems in 2000, and is presently working as an Associate Professor at National University "Lvivska Polytechnika" (Lviv, Ukraine). His interests include the full cycle of embedded systems design together with various processors, operating systems and target applications.
Contact: vkremin@cypressmicro.com

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/