AVR449: Sinusoidal driving of 3-phase permanent magnet motor using ATtiny261/461/861

8-bit AVR® Microcontrollers

Application Note

Features

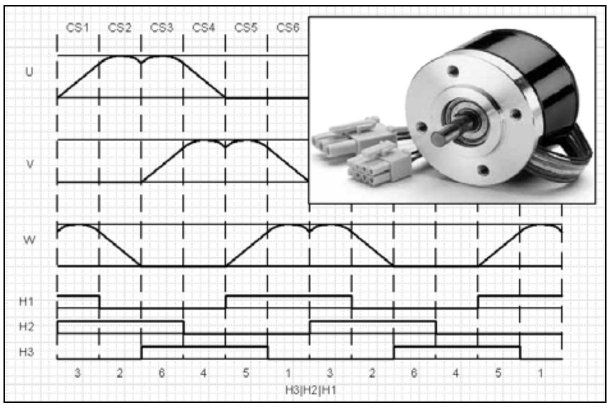
- **Three-phase sine waves**
  - **192 steps per electrical revolution**
  - **10-bit amplitude resolution**
- **Hardware dead-time generation**
- **Controlled by hall-sensors**
- **Speed control through run-time scaling of sine wave amplitude**
  - **Speed reference from analog input**
- **Automatically synchronizes to a running motor at startup**
- **Safe startup using block-commutation the first commutation steps**
- **Direction controlled by digital input**
- **Safe stop and direction change procedure**
  - **Active braking or coasting during stopping**
- **Advance commutation angle adjustable at run-time**

1 Introduction

This application note describes the implementation of sinusoidal driving for three-phase brushless DC motors with hall sensors on the ATtiny261/461/861 microcontroller family.

The implementation can easily be modified to use other driving waveforms such as sine wave with third harmonic injected.

Figure 1-2. Sine wave driving of brushless DC motor with hall sensors.



Rev. 8030B-AVR-10/07

2 Theory of operation

In most of the literature, permanent magnet motors are divided into two categories based on the shape of the back-EMF (voltage induced in the coils when the motor is spinning). The back-EMF can either be trapezoidal or sinusoidal in shape. Although the terminology is not consistent throughout the literature, the majority seems to agree that a brushless DC motor (BLDC) has trapezoidal back-EMF, while a permanent magnet synchronous motor (PMSM) has sinusoidal back-EMF. Both BLDC and PMSM motors can be driven by sinusoidal currents, so there will not be made any distinction between them throughout this application note. Instead, they will both be referred to as a permanent magnet motor, or PMM.

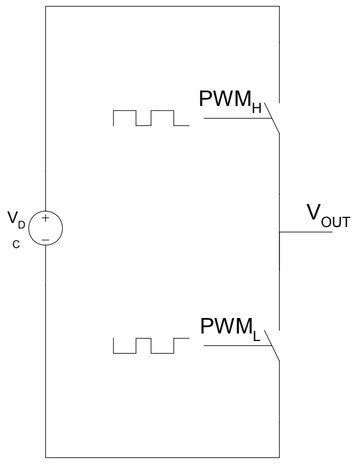
3 Implementation on ATtiny261/461/861

This application note describes how to drive a three-phase PMM with sinusoidal currents. The code example can also be used as a general reference on how to generate waveforms using the PWM.

3.1 Voltage generation

In order to drive a three-phase motor with sinusoidal currents, independent voltages for each phase must be generated. The driver stage for a three-phase motor usually consists of three half-bridges, one for each motor terminal. Each half-bridge consists of two switches, e.g. two power MOSFET transistors. To understand how the phase voltages are generated, it is sufficient to look at one half-bridge. Figure 3-1 shows one half-bridge connected to a DC voltage source.

Figure 3-1. Voltage generation using a half-bridge.



3.1.1 PWM

The average voltage of the output, V_{OUT}, can be regulated between 0V and V_{DC} by applying two inverted pulse-width modulated (PWM) signals to the two switches, PWM_H and PWM_L. The average output voltage will be proportional to the duty cycle of the high side switch. The output, V_{OUT}, will in this case not be a smooth voltage curve, but a square wave similar in shape to the PWM signal applied to the high side switch. If this signal were fed through a low-pass filter, the output voltage would be a voltage level proportional to the duty cycle of the high side switch.

For several reasons, it is not common to add a separate low-pass filter in motor control designs. First of all, the motor acts as a low pass filter. The inductance and resistance of the coils windings create a low pass RL filter. Further, the inertia of the rotor and load creates a mechanical low pass filter. Choosing the PWM switching frequency sufficiently high, there will be no noticeable jitter in the rotor speed. Secondly, the currents fed through the windings of even a small motor can be in the range of several amperes. Forcing this current through e.g. an RC filter would result in substantial power dissipation in the filter itself, an undesirable energy loss.

3.1.2 Dead-time

Switching devices, such as MOSFET transistors are not able to switch on and off instantly. Consider again the half-bridge from Figure 3-1. If the switches PWM_H and PWM_L are fed with inverted signals, one switch will turn off at the same moment as the other switch turns on. During this transition, there will be a short time period where one switch has not completely closed while the other one is opening, making a direct connection between supply voltage and ground with very low resistance, allowing a large current to flow through the transistors. This situation is known as a shoot-through, and must be avoided, since it is likely to overheat and thereby destroy the driver stage if no hardware protection is in place.

The solution to this is to add a dead-time, a small time period where neither the high or low side switches are conducting, for every PWM transition.

3.1.3 Generating PWM signals with dead-time with ATtiny261/461/861

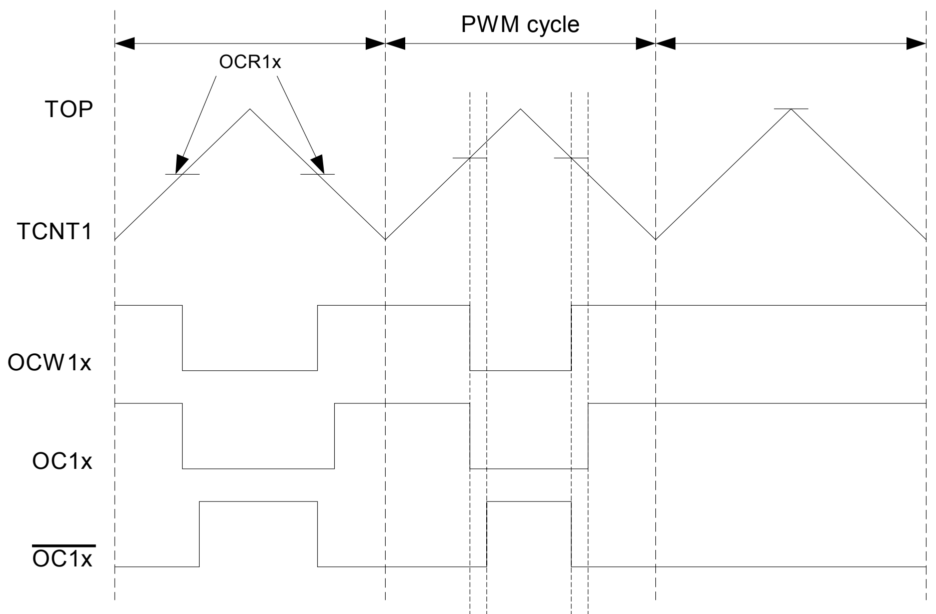
The Timer/Counter1 (TC1) module of the ATtinyX61 family is very well suited for driving three phase motors. It can be run from a 64MHz PLL, and has a resolution of 10 bits. Six PWM outputs can be generated from this timer, grouped into three complementary output pairs with configurable dead-time. The “PWM6” mode is perfect for block commutation of brushless DC motors and a hardware fault protection unit can disable the PWM drivers without any intervention from the CPU for maximum safety.

The “Phase and frequency correct PWM mode” of TC1 is perfect for half-bridge control. In this mode, three pair of complementary PWM outputs with hardware dead-time insertion can be generated. This is exactly what is needed to generate three-phase sinusoidal drive waveforms with a triple half-bridge driver stage.

The operation of one of the three complementary output pairs can be seen in Figure 3-2. The counter counts in an up-down counting mode. The OCR1x register specifies the current duty cycle. An intermediate waveform, labeled as OCW1x in the figure, is generated by clearing the output on a compare match when up-counting and set the output on compare match when down-counting. This intermediate waveform is fed to the dead time generator, which in turn generates two outputs from this waveform. The non-inverted output, OC1x, follows OCW1x, except it will not go high until a specified dead-time period, DT1H, has elapsed after OCW1x goes high. The inverted output, $\overline{\text{OC1x}}$, follows the inverted OCW1x signal, except it will not go high until a specified dead-time period, DT1L, has elapsed after OCW1x goes low. The result is a pair of complementary outputs with dead-time.

To control the duty cycle of one half-bridge, only one register, OCR1x needs to be changed. The average voltage output of the half-bridge will be proportional to this signal.

Figure 3-2. Generating complementary PWM signals with dead-time.



3.1.4 PWM base frequency

The PWM base frequency is controlled by the Timer/counter resolution and clock frequency.

The Timer/Counter1 resolution can be controlled by setting the TOP value in OCR1C. When operated in phase and frequency correct PWM mode, the counter counts up to the TOP value and down to zero during each PWM cycle. One PWM cycle period is $2 \cdot TOP - 2$ clock cycles. In this application note, the full 10 bit resolution of Timer/Counter1 is utilized, giving a PWM cycle period of 2046 timer clock cycles.

Timer/Counter1 is clocked from the system clock or the fast peripheral clock running at a nominal speed of 64 or 32MHz. In this application note, a clock frequency of 64MHz has been used to ensure a PWM frequency well above the audible frequency spectrum. The frequency of the Timer/counter1 clock will be referred to as $f_{CLK,T/C1}$ in the following.

The PWM frequency as a function of Timer/Counter1 clock frequency can be calculated from Equation 3-1.

Equation 3-1. PWM base frequency as a function of timer clock frequency.

$$f_{PWM} = \frac{f_{CLK,T/C1}}{2 \cdot TOP - 2}$$

The PWM frequency with a Timer/Counter1 clock frequency of 64 MHz and 10 bit resolution is thus 31,28kHz.

3.2 Waveform generation

Using the information from section 3.1 on how to set up the AVR® to generate a voltage output from a half-bridge, it is possible to set up the system to drive a motor with a triple half-bridge.

Each half-bridge is connected to each of the 3 independent complementary PWM output pairs of Timer/Counter1. The connections are shown in section 5.

3.2.1 Scaling the waveforms

The needed waveforms are stored in look-up tables (LUTs) to avoid computationally heavy operations. The waveforms are stored at maximum amplitude to utilize the full resolution of the LUT. It is not desirable to output the waveforms at maximum amplitude all the time, so the waveforms need to be scaled before they are output.

Equation 3-2 shows how the output value can be scaled. The output duty cycle, d_o, is obtained by multiplying the output value obtained from the look-up table, d_{table}, with a scaling factor, k_s.

Equation 3-2. Amplitude scaling equation

$$d_o = \frac{k_s d_{table}}{2^n}$$

The output bit resolution can be calculated as $n_o = n_k + n_t - n$, where n_o, n_k and n_t are the bit resolutions of the output, scaling factor and table value respectively and n is the division exponent in Equation 3-2. E.g. a table value resolution of 8 bits, a scaling factor of 10 bits and a division exponent of 8 bits will generate an output duty cycle value with a bit resolution of 10 bits.

Since the ATtinyX61 does not include hardware multiply or divide, calculating Equation 3-2 can be a CPU intensive task. To speed up this calculation, a software multiply and divide routine is included with this application note that multiplies an 8 bit number with a 15 bit number and returns a scaled 15 bit result, discarding the 8 least significant bits of the result. The equation describing the function is expressed in Equation 3-3. Using a 10 bit scaling factor as argument for m_{15} and the 8 bit table value for m_8, results in a 10 bit result, effectively implementing Equation 3-2 with n_k=10, n_t=8 and n=8. The software multiply function has a fixed execution time of 50 clock cycles.

Equation 3-3. Software multiplier equation

$$r = \frac{m_{15} \cdot m_8}{2^8}$$

3.3 Generating sine waves

Section 3.2 explains how to produce an arbitrary waveform stored in a look-up table. In this section, an efficient way to produce sine wave output is explained.

3.3.1 The output pattern

The most straightforward approach would be to just store a sine wave in a look-up table and use this table to generate a sine wave on each motor terminal. However, for motor driving there are more efficient ways to produce sinusoidal drive waveforms. The key to understanding this is that we are not trying to generate sinusoidal signals for each motor terminal with respect to ground. What we are trying to generate are three sinusoidal line-to-line voltages (differential voltage between two terminals) with a phase shift of 120^o between them. Table 3-1 and Table 3-2 shows how this can be accomplished without producing full sine waves for each terminal. A graphical representation along with a typical block commutation pattern is shown in Figure 3-3.

Table 3-1. Terminal and line-to-line voltages, forward driving.

Step	U	V	W	U-V	V-W	W-U
S1-S2	sin(θ)	0	-sin(θ-120)	sin(θ)	sin(θ -120)	sin(θ -240)
S3-S4	-sin(θ-240)	sin(θ -120)	0	sin(θ)	sin(θ -120)	sin(θ -240)
S5-S6	0	-sin(θ)	sin(θ -240)	sin(θ)	sin(θ -120)	sin(θ -240)

Table 3-2. Terminal and line-to-line voltages, reverse driving.

Step	U	V	W	U-V	V-W	W-U
S1-S2	sin(θ)	-sin(θ-120)	0	-sin(θ-240)	-sin(θ-120)	-sin(θ)
S3-S4	-sin(θ-240)	0	sin(θ -120)	-sin(θ-240)	-sin(θ-120)	-sin(θ)
S5-S6	0	sin(θ -240)	-sin(θ)	-sin(θ-240)	-sin(θ-120)	-sin(θ)

There are two advantages to this approach. First of all, the maximum line-to-line voltage generated is higher than with the pure sine wave approach, offering higher torque and speed. Secondly, each terminal output is zero for 1/3 of the time, reducing switching losses in the power stage. A side effect of this is that the scaling in Equation 3-2 only needs to be performed twice for each output update, since the result of one of the scaling multiplications is always zero.

3.3.2 Organizing the look-up table

Organization of the look-up table is a trade-off between access time and look-up table size. Looking at the waveforms in Figure 3-3, it can be seen that the information can be "compressed" in several ways.

The three waveforms are phase shifted by 120 degrees, so it is possible to only store one waveform and use that for all three terminal voltages. Furthermore, looking at the waveform for phase U, it can be seen that steps S3-S4 is a mirrored version of S1-S2. In S5-S6, U is simply zero. This suggests that it is possible to get away with storing as little as one third of the table and using software to find the correct value.

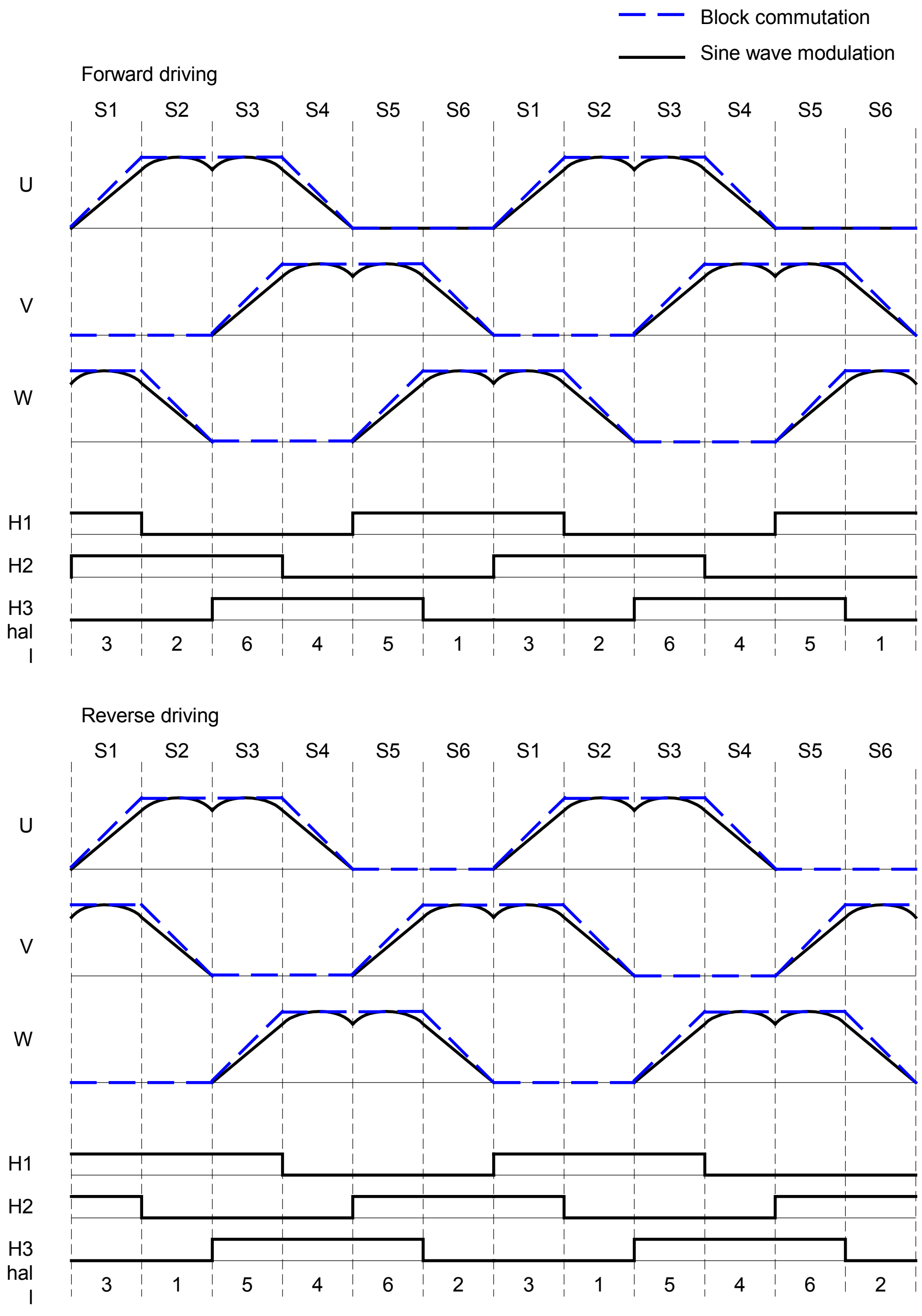
Since the table lookup must be performed for all three channels for every PWM cycle, even a small overhead can contribute significantly to average CPU load. For maximum performance it is possible to store the waveforms for all three terminal voltages in the look-up table. To ensure the fastest possible access, the values are organized as shown in Table 3-3. This allows all 3 values to be read with the very fast `LPM Z+` instruction. To get the output values for forward driving, the table is read in the sequence U, V, W, while in the reverse direction, the sequence U, W, V is used.

Table 3-3. Sine look-up table organization.

U_0	U_0 - 120^o	U_0 - 240^o	U_1	U_1 - 120^o	U_1 - 240^o	...

This is the preferred method for storing the table. However, for smaller devices, like the ATtiny261, a table like this takes up quite a large percentage of the available program memory. For this reason, a small table (one third of one phase) version is included with this application note. This version uses approximately 25% more CPU cycles when updating the PWM outputs, so the CPU load will be larger using this version. It is possible to select the smaller version by changing configuration parameters before compiling.

Figure 3-3. Sine wave generation.



3.4 Timing

The ATtinyX61 has an additional 8/16 bit timer that is left unused for other purposes in this application note. It can for example be used to measure the duty cycle or frequency of an external digital speed reference signal. Timing related to generation of sine waves is instead performed using the overflow interrupt of the Timer/Counter1 units as a time base. The time period between each overflow interrupt will be referred to as a ‘tick’. The duration of one tick is equal to the PWM time period, which is given by Equation 3-4.

Equation 3-4. PWM time period.

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{2 \cdot TOP - 2}{f_{CLK,T/C1}}$$

As an example, with 64MHz Timer/Counter1 clock frequency and 10 bit resolution, one tick has a duration of 31,97µs.

3.5 Position sensors and their usage

Since the objective is to control a synchronous motor, some kind of information about the rotor position must be known in order to produce waveforms that are synchronized to the motor. This information can be obtained, e.g. by using a rotary encoder. However, many permanent magnet motors are equipped with three hall sensors mounted providing information about the rotor positions in 60 degree increments. In this application note, the hall sensors are used as the only position sensing devices.

In Figure 3-3, the three hall sensor signals are shown as H1, H2 and H3. The hall sensor inputs translate into a number between 1 and 6 by treating each of the hall sensor signals as binary digits and arranging them as *hall = H3 H2 H1.* The *hall* values corresponding to the different combinations of *H3*, *H2* and *H1* are shown in Figure 3-3.

The state of the hall sensors and the timing of hall sensor changes are used for several purposes:

- Output waveform phase locked loop
- Speed calculation
- Block commutation
- Rotation detection.
- Synchronization and direction change
- Advance commutation angle control

3.5.1 Phase locked loop

The goal of the waveform generation is to keep the output waveforms synchronized to the rotation of the rotor. The challenge when using hall sensors as position sensors is that up-to-date information about rotor position is only available every 60 degrees. To solve this challenge, a phase locked loop must be implemented that keeps the output waveform in synch with the rotor.

An index (or pointer) into the look-up table is maintained at all times. The information obtained from the hall sensors is used to update this index.

The most accurate position information is available in the exact moment when a hall sensor changes value. At this point, the exact angle of the rotor is known.

In the time between hall sensor changes, no information about the position of the rotor is available. However, we do know the position of the rotor at the last hall sensor change, and we can measure the time period between the last two hall sensor changes. This allows us to calculate the speed of the rotor and, assuming constant speed, the position can be calculated from Equation 3-5.

Equation 3-5. Position interpolation.

$$\dot{\theta} = \omega$$

$$\theta(t) = \theta_0 + \omega t$$

where θ is the angular position, θ_0 is the angular position at the last hall sensor change, ω is the angular velocity and t is the time since the last hall sensor change.

Without going into detail, Equation 3-5 can be approximated by the difference equation shown in Equation 3-6.

Equation 3-6. Angular displacement difference equation.

$$\theta[k] = \theta[k-1] + \omega T \text{ ,}$$

where θ is angular position, k is the current time step, ω is the angular velocity and T is the time step size.

In this application note, the angular position is represented by the look-up table index, measured in table steps. Time is measured in ticks. Angular velocity is defined by Equation 3-7.

Equation 3-7. Angular velocity definition

$$\omega = \frac{\Delta\theta}{\Delta t} \text{,}$$

where $\Delta\theta$ is the angular displacement during the time period Δt. Since angular displacement is measured in table steps and time is measured in ticks, the angular velocity unit is table steps per tick.

The look-up table index is updated once every tick, so the time step, T, of Equation 3-6 is 1 tick. The increment, used to iterate the table between two consecutive hall changes can thus be calculated from Equation 3-8.

Equation 3-8. Look-up table index increment calculation.

$$i = \omega T = \frac{\Delta\theta}{\Delta t} T = \frac{n_e}{n_T} \text{,}$$

where n_e is the number of table elements per electrical revolution and n_T is the number of ticks taken to rotate one electrical revolution.

At every tick, the increment is added to the look-up table index to update the position information.

The algorithm used to implement the phase locked loop can thus be summarized as:

1. At every hall sensor change:
   - Set the look-up table index to correspond with the rotor position.
   - Calculate the index increment from Equation 3-8.
2. For every tick until next hall sensor change:
   - Update the look-up table index using Equation 3-6.
   - Update PWM duty cycles according to look-up table index.
3. When hall sensors change, go back to 1.

3.5.2 Speed calculation

If closed loop speed control is needed, the rotational speed of the motor must be calculated. As explained in section 3.5.1, the speed information is already calculated in the form of a look-up table increment value. There is no need to perform yet another computationally intensive calculation to obtain information about the speed. Nor is it necessary to have yet another variable with information about speed. Resources can be saved by representing other speed values, such as speed controller set-point in the same unit as the increment, since it corresponds directly to the speed information already available.

To control the speed to a certain RPM value, this value must first be converted to the corresponding increment value. It is therefore necessary to know the relationship between rotational speed and increment. Speed in revolutions per minute (RPM) is related to the number of ticks between reading the same hall sensor value twice, as shown in Equation 3-9.

Equation 3-9. RPM calculation.

$$\omega_{RPM} = 60 \frac{f_{CLK,T/C1}}{(2 \cdot TOP - 2) \cdot n_T}$$

Rearranging to give ticks per electrical revolution as a function of speed:

Equation 3-10. Ticks between hall changes as a function of speed.

$$n_T = \frac{60 \cdot f_{CLK,T/C1}}{(2 \cdot TOP - 2) \cdot \omega_{RPM}}$$

Combining Equation 3-8 and Equation 3-10 gives index increment as a function of speed (RPM):

Equation 3-11. Increment as a function of speed in RPM.

$$i = \frac{n_e \cdot (2 \cdot TOP - 2)}{60 \cdot f_{CLK,T/C1}} \omega_{RPM}$$

3.5.3 Block commutation

During the start-up phase, the speed of the rotor is not known until two subsequent hall sensor changes has been detected. To ensure a robust start-up, block commutation is used until the rotor speed is known. The principles of BLDC control using block commutation will not be covered in this application note. For more information, see e.g. application note AVR443.

Block commutation can be implemented using the Timer/Counter1 PWM6 mode on the ATtinyX61. All outputs are operated at the same duty cycle, controlled by a single output compare register, OCR1A. Another register, TCCR1E, controls which PWM outputs are enabled at any time. TCCR1E is changed every time there is a hall sensor change. One table for each direction is used to hold the output patterns for the corresponding hall sensor inputs. The output pattern used in block commutation mode with respect to hall sensor input is also illustrated in Figure 3-3 along with the sinusoidal pattern.

3.5.4 Rotational direction detection

The sequence of hall sensor changes can be used to determine the actual direction of rotation. One table for each direction is used to store the next expected hall sensor value for each hall sensor value. This is used to deduce the actual direction of rotation.

Comparing the actual direction of rotation to the commanded direction of rotation determines if the motor is spinning in the desired direction.

3.5.5 Synchronization and direction change

When the microcontroller is powered up or reset, the motor might already be spinning. It is therefore important that the firmware determines the state of the motor and synchronizes to this state.

When changing the direction of rotation, the rotor must first be stopped before it can be driven in the opposite direction. If the direction command is changed once again before the motor has stopped, the firmware must resynchronize before applying voltage to the terminals.

Synchronization and turning is controlled by the hall sensors. Direction information is deduced as described in section 3.5.4.

The motor is considered to be stopped when a sufficient number of ticks have passed since the last hall sensor change.

Synchronization occurs when the motor is spinning in the commanded direction and at least two hall sensor changes are detected.

3.5.6 Advance commutation control

A programmable advance commutation angle can be used to adjust the phase of the output waveform to make it lead the rotor angle. This might be necessary to get the motor to run at it's maximum speed and/or efficiency. The advance commutation angle is run-time adjustable and can be set up to vary according to e.g. rotor speed or waveform amplitude. The advance commutation angle is obtained by adding an offset to the look-up table index, thus shifting the phase of the waveforms. The advance commutation angle can be adjusted in increments of 1 look-up table step (1.8˚).

3.6 Overcurrent detection

A locked rotor, sudden load change or fast acceleration can cause excessive current to run through the motor and driver stage. In order to prevent damage due to overcurrent, it is very important to monitor current at all times.

The Timer/Counter1 unit on the ATtinyX61 has a fault protection unit that can disable all Timer/Counter1 PWM outputs without intervention from the CPU in case of an overcurrent or other emergency situation. It will also trigger an interrupt, so the CPU is made aware that the fault protection has been triggered. The fault protection unit can be set up to trigger on a digital external interrupt or by the internal analog comparator.

In this application note, the digital external interrupt has been used as trigger source for the fault protection unit. When the fault protection unit is triggered, the application waits for a while and then restarts the motor. Both the source for the fault protection unit and the correct action for a fault protection event will differ from application to application, so it is recommended that these parts are changed to fit the design.

3.7 Speed control

This application note includes examples of both open loop speed control and a closed loop PID controller. It is also possible to add other kinds of speed controllers if desired.

3.7.1 Speed reference

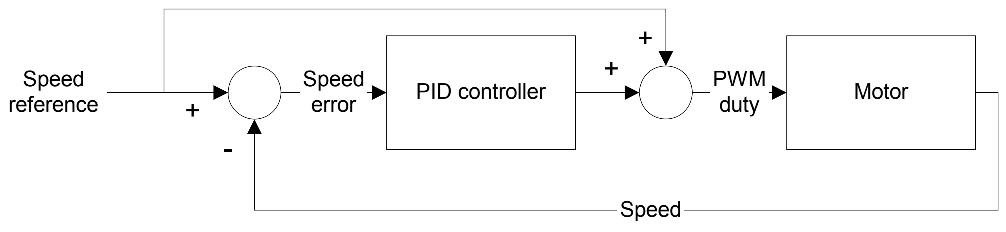
The controller, whether open or closed loop, needs some kind of speed reference. In this application note, an analog voltage reference is used, although it could easily be exchanged with e.g. a UART command or a digital frequency or duty cycle signal. The analog reference is measured using the ADC.

3.7.2 Speed controller

The open loop speed control is very simple. The 10 bit analog speed reference value is directly used as the 10 bit amplitude scaling value for the generated sine waves.

The closed loop speed control also uses the 10 bit analog speed reference value as a feed forward value to the amplitude setting. In addition, a PID (proportional, integral derivative) controller is used to make sure that the speed is accurately controlled to the desired speed. The ADC measurement is used as setpoint for the speed controller. Since the internal representation of speed is index increment, the measured signal must be converted to the same representation. Section 3.5.2 covers the relationship between speed in RPM and the internal index increment representation. A block diagram of the closed loop system with feed forward is shown in Figure 3-4.

Figure 3-4. PID controller with feed forward.



The speed control loop is the only part of the motor control application that is not interrupt driven. This is because the PID calculations take too long to perform inside an interrupt routine without degrading the performance of the motor control. Furthermore, it is not necessary to run the control loop as often as each commutation.

4 Firmware implementation

The source code included with this application note is fully documented with Doxygen comments which explains all parts of the code. The full Doxygen documentation in html format can be accessed by opening the 'readme.html'.

This chapter includes additional information needed to understand the overall flow of the implementation.

4.1 Code structure

Note that the code included with this application note has been written for high performance. Because of this, almost all source code is contained in one file to allow the compiler to optimize the code as much as possible. Most functions are declared with the "`#pragma inline=forced`" directive, since they are called from interrupt routines.

4.2 Peripheral usage

The hardware peripherals in the ATtinyX61 used in this application note are listed in Table 4-1.

Table 4-1. Hardware module usage.

Hardware module	Usage
Timer/counter1	PWM modulation
External interrupt 0	Emergency shutdown
ADC	Speed reference input
Pin change interrupt 1	Hall sensor change interrupt

4.3 Actions performed in interrupts

The full motor control application, except speed control, is interrupt-based. Table 4-2 shows the responsibility of each interrupt used in the application. Understanding the responsibility of each interrupt service routine is the key to understanding how the application works.

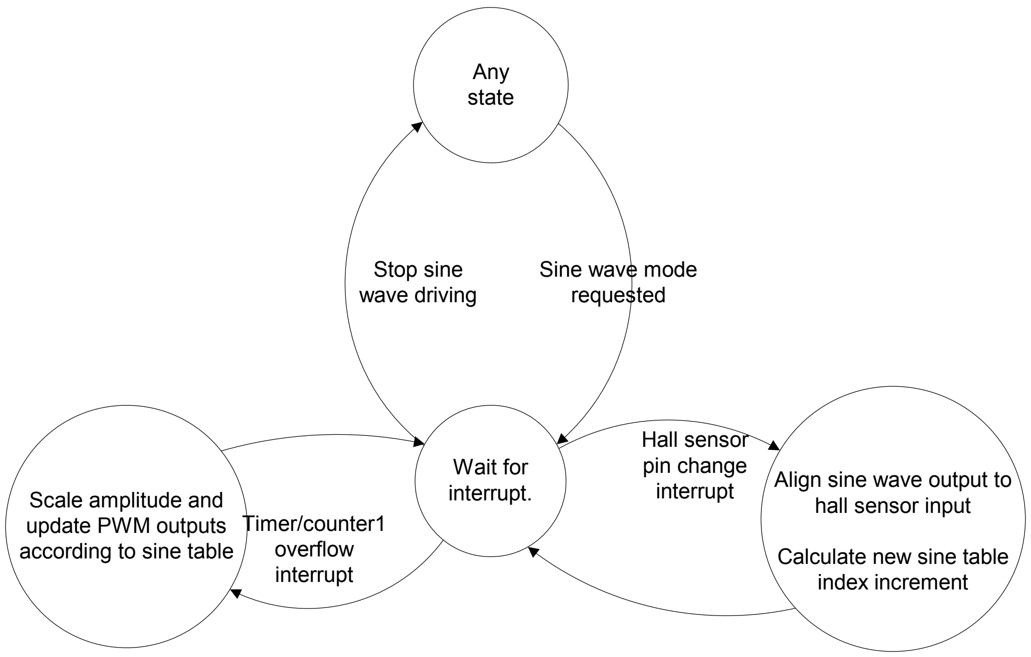
Table 4-2. Interrupt responsibility.

Interrupt	Function	Responsibility/action
Timer/counter1 overflow	`Timer1OverflowISR`	PWM compare value update. Block commutation duty cycle control. Speed measurement. Stop detection. Direction control input.
Pin change 1	`HallChangeISR`	Synchronizes sine wave to hall sensors. Block commutation. Actual direction detection. Detects whether firmware and motor are synchronized. Sine table index increment calculation. Stop detection.
Fault protection	`FaultProtectionISR()`	Fault protection.

4.4 Output waveform generation

Two interrupt service routines cooperate to produce the sine wave output. `Timer1OverflowISR` updates the PWM compare values once every PWM cycle. `HallChangeISR` synchronizes the sine wave to the current rotor position and calculates the sine table index increment. Figure 5-2 shows the interaction between the two ISRs. The state labeled ‘Any state’ symbolizes that it is not relevant what state the motor was in or goes to.

Figure 4-1. Sine wave generation state machine.


Any state
Stop sine wave driving
Sine wave mode requested
Wait for interrupt.
Hall sensor pin change interrupt
Align sine wave output to hall sensor input
Calculate new sine table index increment
Scale amplitude and update PWM outputs according to sine table
Timer/counter1 overflow interrupt

4.5 Direction and synchronization control

4.5.1 Related flags

Two definitions are needed to explain the direction and synchronization control:

Synchronized means that a specified number of subsequent hall sensor inputs corresponds to the pattern expected when rotating in the direction commanded by the direction input pin. This is used to ensure that sine wave driving is applied with the correct frequency, phase and direction.

Stopped means that there has not been a change in any of the hall sensor inputs for a specified number of ticks.

Two flags that are part of the global `fastFlags` variable indicates whether the motor is currently synchronized and/or stopped. These flags are automatically manipulated by the interrupt service routines at certain events. The following functions/ISRs modify these flags:

`CommutationTicksUpdate`:

- motorStopped = TRUE, if a predefined number of 'ticks' has passed since last hall sensor change.
- motorSynchronized = FALSE, if motorStopped flag has just been set to TRUE.

This function is called by `Timer1OverflowISR`.

`MotorSynchronizedUpdate`:

- motorSynchronized = TRUE, if the synchronized criteria is met.

This function is called by `HallChangeISR`

`HallChangeISR`:

- motorStopped = FALSE (motor can not be stopped if the hall sensors change value)

`Timer1OverflowISR`:

- motorSynchronized = FALSE, if the desired direction has changed.

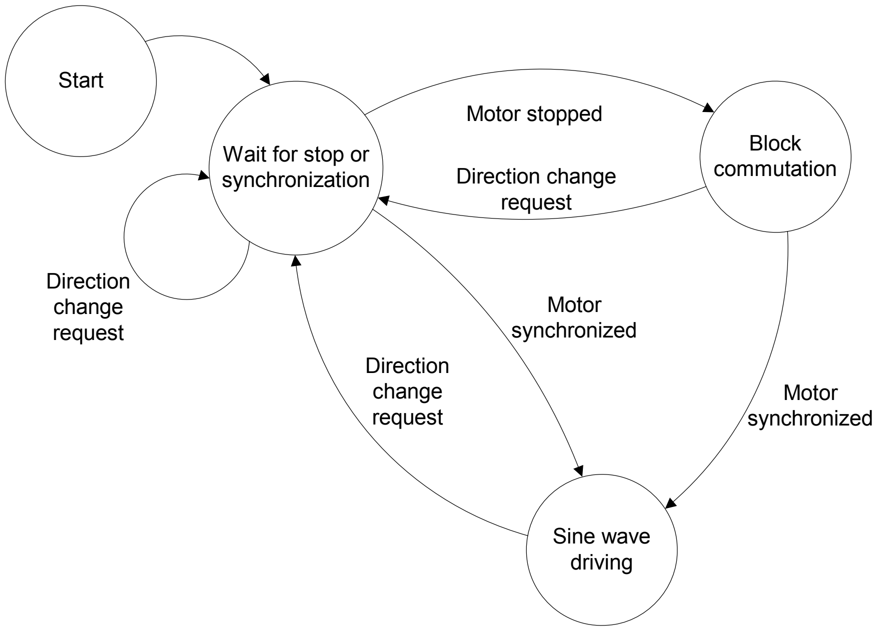
4.5.2 Direction and synchronization logic

There are several situations where the motor control firmware needs to synchronize to the motor before any waveform is applied:

- When the motor is started from standstill, it is first commutated using block commutation. Block commutation is used until synchronization is obtained. This ensures that sine waves with correct frequency and phase are generated when the motor switches to sine wave driving.
- When the microcontroller is started and the motor is already running, the firmware will not apply any driving waveform until it is synchronized to the motor, or the rotor has stopped turning.
- When a direction change is requested, driving will be disabled or braking initiated until the motor is stopped. If another direction change is requested, the motor may be able to synchronize again. In that case, it resumes sinusoidal driving at the correct frequency without waiting for the motor to stop.

The full direction and synchronization control is illustrated in Figure 4-2. Motor stopped means that the `motorStopped` flag is TRUE. Motor synchronized means that the `motorSynchronized` flag is TRUE. Direction change requested means that the direction input has changed since last time it was checked.

Figure 4-2. Direction and synchronization control state machine.



5 Hardware

This chapter describes how to connect the hardware for use with this application note.

5.1 Pin assignment

The pin assignment used in this application note is shown in Figure 5-1. The function of each pin is described in Table 5-1.

Figure 5-1. Pin assignment.

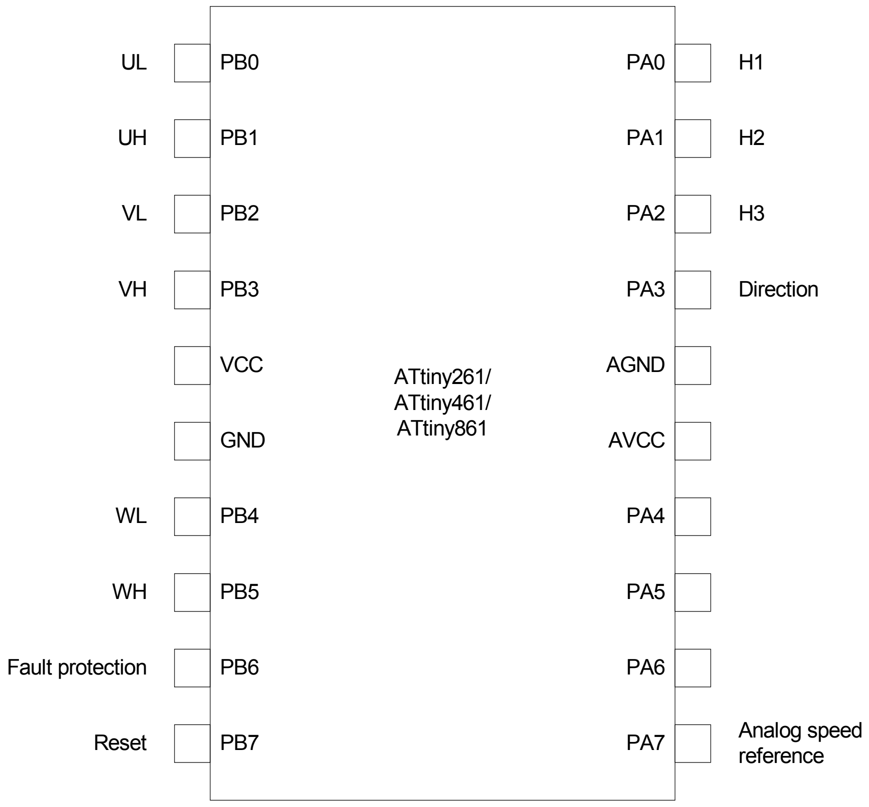


Table 5-1. ATtiny261/461/861 pin usage.

Pin	Name	Purpose	Direction
PA0	H1	Hall signal 1	In
PA1	H2	Hall signal 2	In
PA2	H3	Hall signal 3	In
PA3	Direction	Low = forward, high = reverse.	In
PA7	Speed reference	Analog level controlling amplitude of waveforms. Expects signals in the range 0-2.56V.	In
PB0	UL	Phase U low side control signal	Out
PB1	UH	Phase U high side control signal	Out

Pin	Name	Purpose	Direction
PB2	VL	Phase V low side control signal	Out
PB3	VH	Phase V high side control signal	Out
PB4	WL	Phase W low side control signal	Out
PB5	WH	Phase W high side control signal	Out
PB6	Fault protection	When this pin goes low, the PWM outputs are disabled and the fault protection ISR is entered.	In
PB7	Reset	Resets the microcontroller.	In

5.2 Connecting the ATtiny261/461/861 to a driver stage and motor

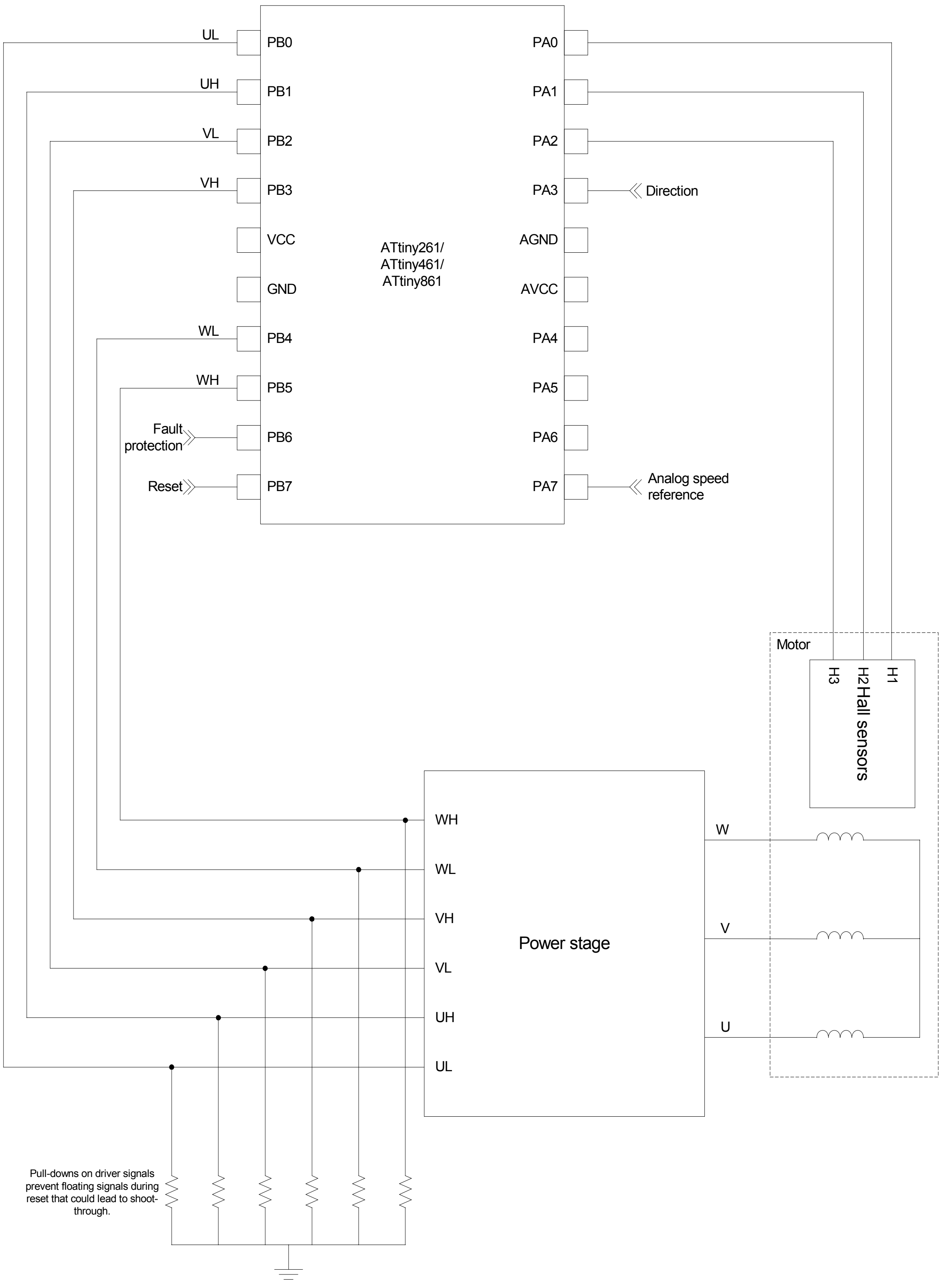
This application note needs following components to work:

- A 3 phase permanent magnet motor with hall sensors.
- A driver stage capable of driving the motor.
- An Atmel® ATtiny261/461/861 microcontroller.
- An analog input signal in the range 0-2.56V (For speed control).

Figure 5-2 shows a conceptual schematic of the full system.

Note that the *Fault protection* and *direction* signals, even if not used, should be connected to a fixed logic level for the included firmware to work correctly.

Figure 5-2. Conceptual schematic of the full system.



5.2.1 Using the ATAVRMC100 driver stage

This application note has been tested on the ATAVRMC100 power stage/motor development kit. The ATAVRMC100 has an onboard AT90PWM3 microcontroller mounted, but it is possible to use it as a power stage with a different microcontroller through the EXT_DRV and SENSOR interfaces on the board. The EXT_DRV header is documented in the ATAVRMC100 Hardware User Guide, available from the Atmel web site. It is important to erase the AT90PWM3 before using the ATAVRMC100 with a different microcontroller to ensure that only one microcontroller drives the signals.

Table 5-2 contains a complete list of the signals that must be connected to the ATAVRMC100 board. Note that the GND signal is connected to the negative shunt terminal, which is actually directly connected to ground. Figure 5-3 shows a graphical representation of the EXT_DRV header.

Table 5-3 shows the connections necessary to interface with the hall sensors. A graphical representation of this interface is shown in Figure 5-4.

Table 5-2. ATAVRMC100 EXT_DRV connections.

Signal name	ATtinyX61 pin	ATAVRMC100 signal	EXT_DRV pin number
UH	PB1	H_A	1
UL	PB0	L_A	2
VH	PB3	H_B	3
VL	PB2	L_B	4
WH	PB5	H_C	5
WL	PB4	L_C	6
GND	GND	V shunt-	8

Figure 5-3. EXT_DRV connection.

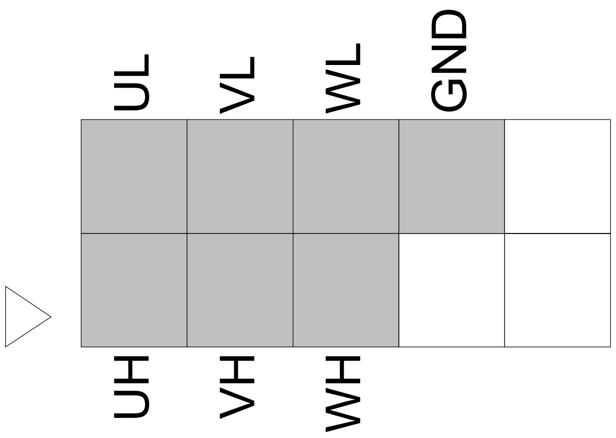
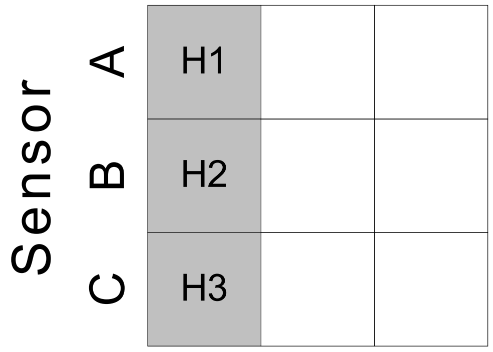


Table 5-3. ATAVRMC100 hall sensor interface.

Signal name	ATtinyX61 pin	ATAVRMC100 signal
H1	PA0	Sensor A
H2	PA1	Sensor B
H3	PA2	Sensor C

Figure 5-4. Hall sensor connection.



6 Waveform plots

Figure 6-1 and Figure 6-2 show the waveforms generated when driving forward and reverse at different speeds. The average function on the oscilloscope has been used to smooth the waveforms. Both plots show, from the top: line-to-line voltage U-V, phase voltages for U, V, W, then hall sensors H3, H2 and H1.

Figure 6-1. Forward driving at approximately 1600 RPM.

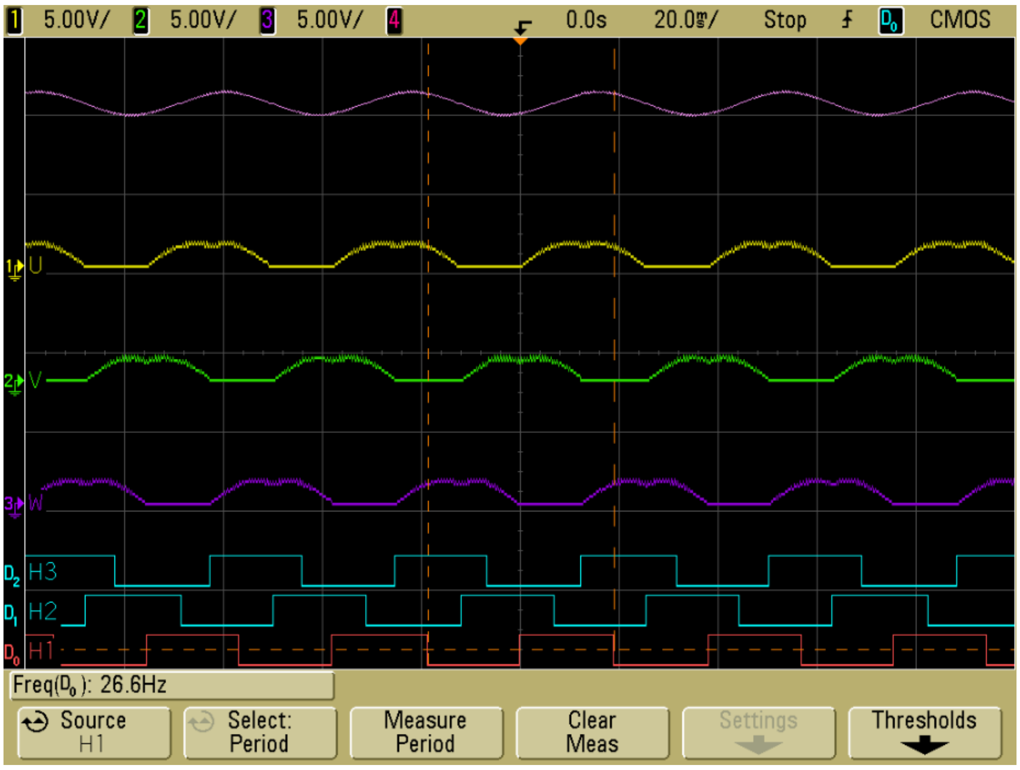
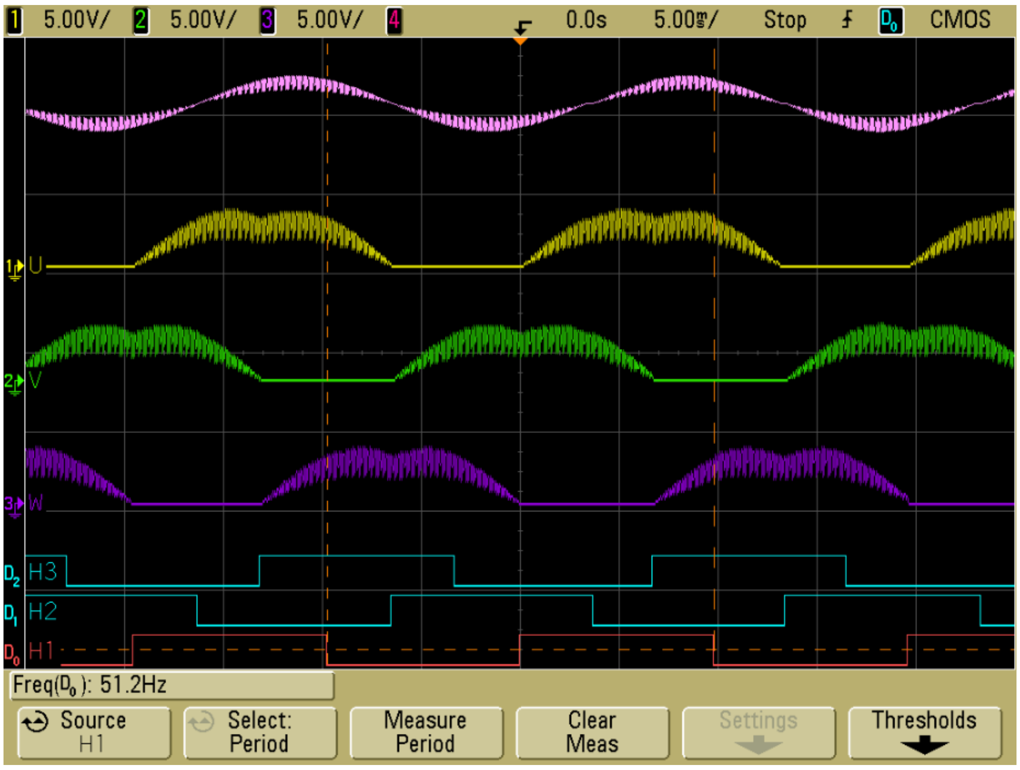


Figure 6-2. Reverse driving at approximately 3000 RPM.



7 Code size and performance

Table 7-1 shows the flash and SRAM memory usage.

Table 7-1. Flash and SRAM usage

Table size	Speed control	Flash	SRAM
Small	Open loop	~1.6kB	~50B
Small	Closed loop	~2.5kB	~70B
Large	Open loop	~2.0kB	~50B
Large	Closed loop	~3.0kB	~70B

8 References

1. Valentine, R., Motor control electronics handbook, 1998, McGraw-Hill.
2. Atmel corporation, ATAVRMC100 hardware user guide rev. B, February 2006 http://www.atmel.com/dyn/resources/prod_documents/doc7551.pdf

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature