AVR32710: Space Vector Modulation using AVR32 UC3 Microcontroller

32-bit AVR® Microcontroller

Application Note

Features

- **Brushless DC Motor**
- **Space Vector Pulse Width Modulation using 6 PWM channels**
- **Source Code for GCC compiler**
- **Use AVR®32 Digital Signal Processing library (DSPLib)**

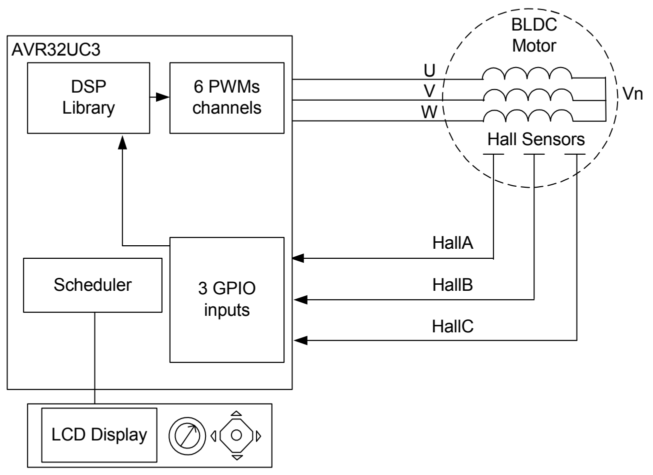
1. Introduction

This application note outlines a demonstration using a stand-alone application on an AVR32 target. It is a real-time system that computes Space Vector Modulation on a Brushless DC Motor.

The Space Vector Modulation technique is described in application note **AVR435** available on www.atmel.com.

This application is designed to work with the EVK1100 evaluation kit, therefore all the following information is specific to this board and this specific microcontroller (AVR32UC3A). Additionally, this application can be easily used by the EVK1101 evaluation kit and the specific microcontroller (AVR32UC3B).

Figure 1-1. Block Diagram



Rev. 32094B-AVR32-05/09

2. Related Parts

This document applies to the following AT32UC3 parts:

- AT32UC3A0512
- AT32UC3A0256
- AT32UC3A0128
- AT32UC3A1512
- AT32UC3A1256
- AT32UC3A1128
- AT32UC3B0256
- AT32UC3B0128
- AT32UC3B064
- AT32UC3B1256
- AT32UC3B1128
- AT32UC3B164

3. Related Items

The software provided with this application note requires several components:

  – **AVR32 GNU Toolchain**: AVR32 GNU Toolchain is a set of standalone command line programs used to create applications for AVR32 microcontrollers (compiler, assembler, linker, debugger). *http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118*

- or

  – **IAR Embedded Workbench® for Atmel AVR32**: IAR Embedded Workbench provides a suite of AVR32 development tools for embedded systems (compiler, assembler, linker, debugger). *http://www.iar.com/*

- **EVK1100**: The EVK1100 is an evaluation kit and development system for the AVR32 AT32UC3A microcontroller. *http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4114*
- **EVK1101**: The EVK1101 is an evaluation kit and development system for the AVR32 AT32UC3B microcontroller. *http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4175*
- **AVR32 UC3 Software Framework**: This framework provides software drivers and libraries to build any application for AVR32 UC3. This is where the DSPLib is located. *http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4192*
- **AT32UC3A0512 Datasheet**: *http://www.atmel.com/dyn/products/product_card.asp?part_id=4117*
- **AT32UC3B0256 Datasheet**: *http://www.atmel.com/dyn/products/product_card.asp?part_id=4174*
- **Space Vector PWM Software Package**: *avr32710.zip*

4. Abbreviations

- **BLDC**: Brushless Direct Current
- **SVPWM** : Space Vector Pulse Width Modulation
- **CW** : Clockwise direction / **CCW** : Counter Clockwise direction
- **FCPU** : CPU frequency
- **FPBA** : Peripheral Bus A frequency

5. Motor Control Theory

5.1 Introduction

The most common method used to control DC motors is to modulate the voltage and especially the duty cycle. This is called the Space Vector Pulse Width Modulation (S.V.P.W.M).

At first, this application note describes the S.V.P.W.M technique and then how to implement BLDC motor control algorithm on AVR32 UC3 devices.

5.2 Space Vector Pulse Width Modulation Principle

5.2.1 Phase Switching

Figure 5-1. Typical structure of the application

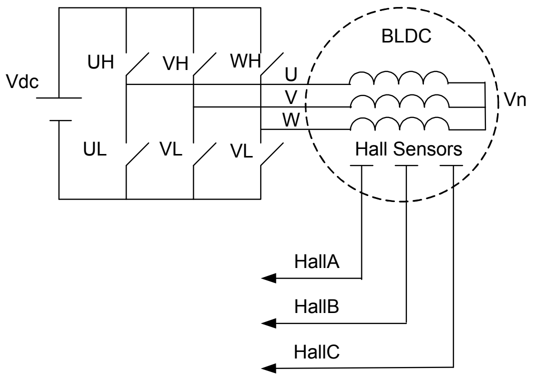


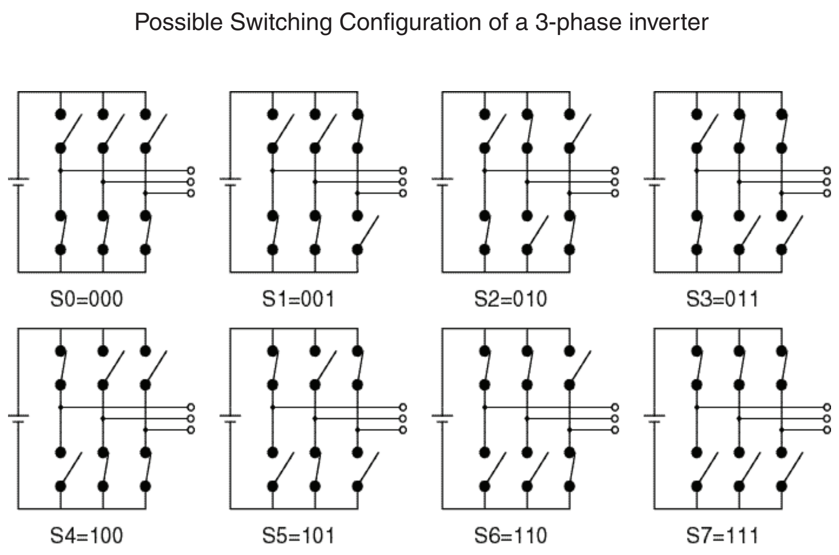
Figure 5-1 shows the typical structure of a BLDC motor connected to a Voltage Source Inverter. Since the motor is considered as a balanced load with an unconnected neutral,

then

$$\overrightarrow{V_{an}} + \overrightarrow{V_{bn}} + \overrightarrow{V_{cn}} = \overrightarrow{0}$$

For this three phase power inverter, there are eight possible switching states.

Figure 5-2. Possible switching configurations of a 3-phase inverter



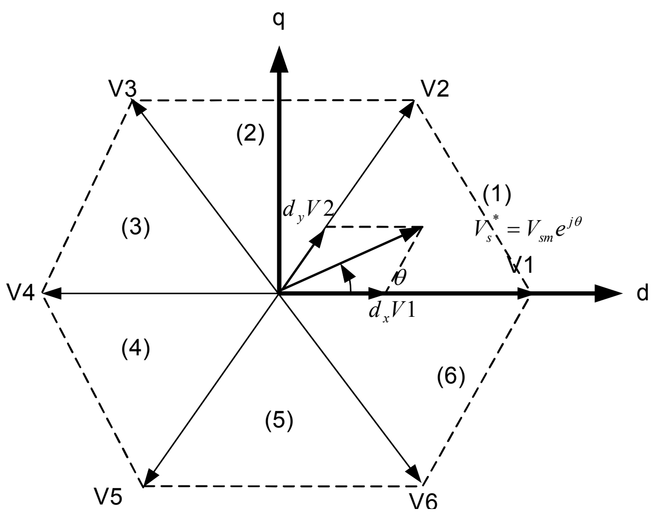
These combinations generates eight 3-phases voltages combinations

Table 5-1. Switching configurations and output voltages of a 3-phase inverter

WH	VH	UH	V_{an}	V_{bn}	V_{cn}
0	0	0	0	0	0
0	0	1	-Vdc/3	-Vdc/3	+2Vdc/3
0	1	0	-Vdc/3	+2Vdc/3	-Vdc/3
0	1	1	-2Vdc/3	-Vdc/3	-Vdc/3
1	0	0	+2Vdc/3	-Vdc/3	-Vdc/3
1	0	1	Vdc/3	-2Vdc/3	Vdc/3
1	1	0	Vdc/3	Vdc/3	-2Vdc/3
1	1	1	0	0	0

5.2.2 New Space Vectors using Clarke Transformation

Figure 5-3. Basic Space Vectors



These 3-phases voltage combinations are converted by Clarke transformation and could be expressed in complex form.

In sector (1), for example,

$$\begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} = K \begin{bmatrix} 1 & -\cos(\frac{\Pi}{3}) & -\cos(\frac{\Pi}{3}) \\ 0 & +\cos(\frac{\Pi}{6}) & -\cos(\frac{\Pi}{6}) \end{bmatrix} \cdot \begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix}$$

So,

$$\begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 - \frac{1}{2} - \frac{1}{2} \\ 0 \frac{\sqrt{3}}{2} - \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix}$$

It implies as well that switching configuration tables could be expressed in (d-q) format

Table 5-2. Switching configurations and output voltages of a 3-phase in (d-q) format

WH	VH	UH	V_{ds}	V_{qs}	Vector	Sector
0	0	1	2Vdc/3	0	V1	(1)
1	1	0	+Vdc/3	+Vdc/3	V2	(2)
0	1	0	-Vdc/3	+Vdc/3	V3	(3)
1	1	0	-2Vdc/3	0	V4	(4)
1	0	0	-Vdc/3	-Vdc/3	V5	(5)
1	0	1	Vdc/3	-Vdc/3	V6	(6)

Each space vector corresponding to WH,VH and UH could be expressed as a combination of Vds and Vqs (vector list in Figure 5-2). This mean for example (WH,VH,UH)=(1,0,0)=V5

5.2.3 Stepping increment and sampling frequency

In this case, it is possible to define any stator voltage as:

$$V_s = V_{\max} e^{\theta} = dxU_x + d_y U_y$$

With

$$dx = \sin\left(\frac{\Pi}{3} - \theta\right) \qquad (1)$$
$$dy = \sin(\theta) \qquad (2)$$

As shown in Figure 5-3, the angle decomposes every step of Space Vector computation algorithm. As the expression of stator voltage is identical in all the 6 sectors, it is possible to only compute equation (1) and (2).

But, every step in computation is given for a angular frequency (in cycles/sec) :

$$\theta(n) = \frac{\Pi}{3} \times \frac{Ftick}{2^{resolution}} \times n$$

The Ftick is defined as the sampling frequency or the basic frequency of the application. n is the angle stepping increment from the range

$$\left[0; \frac{Ftick}{2^{resolution}}\right]$$

For example, with a frequence of tick equal to 20KHz and a resolution up to 8 bits,it is possible to define a step range from [0;78] thus an angle precision of 0.77 degree.

As the UC3 family has DSP instructions support, the sinus and cosinus computation will be done in software for SVPWM and not with a lookup table. Thus, it is possible to define a specific resolution for an application.

6. Motor Control Processing computations

6.1 Flow Chart Diagram

From the SVPWM technique, it is possible to distinguish several actions:

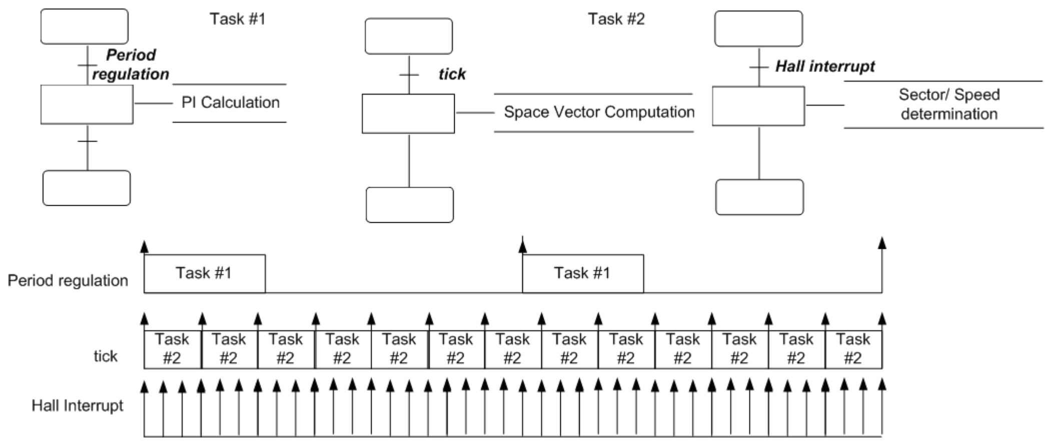
- Sector Determination
- PI Corrector
- Space Vector Computation

Before starting application development, it could be interesting to consider clock source reference:

- Sector Determination: at every hall sensors interrupt a new sector position is reached. So that means that this interrupt should trigger a task that compute Sector Determination Algorithm (electrical position of stator).
- Space Vector Computation

Here is the flow chart diagram for the process in this application.

Figure 6-1. Flow chart diagram



Task #1

Every ‘Period regulation’ period the corrector (Proportionnal and Integer: P.I) is executed and updates amplitude factor used by the S.V.P.M.

Task #2

Every ‘tick’ period in the Space Vector calculation is computed.

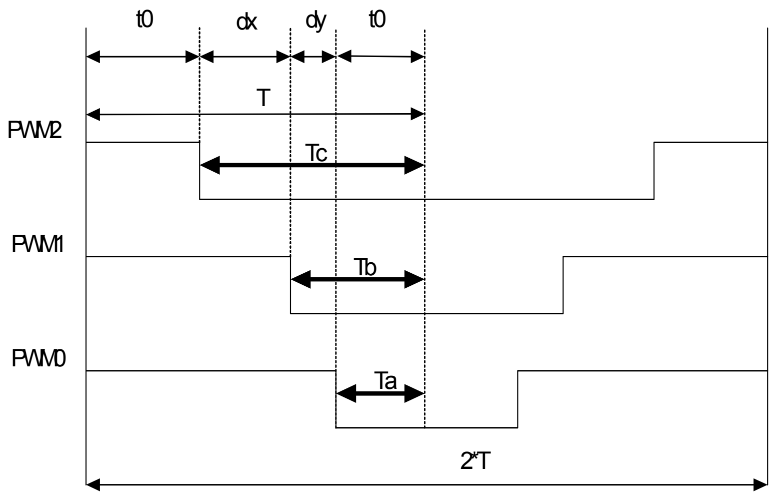
6.2 Space Vector Modulation

For an efficient implementation of SV-PWM, we use DSP library for cosinus and sinus computation.

$$dx = \sin(\frac{\Pi}{3} - \theta)$$
$$dy = \sin(\theta)$$

Once dx and dy are calulated, a set of 3 compare values Ta,Tb and Tc need to be calculated every PWM period to generate this pattern.

Figure 6-2. PWM output switching pattern case in Sector 1



We can say that for this sector 0:

$$T_a = \frac{(T - d_x - d_y)}{2}$$
$$T_b = d_x + T_a$$
$$T_c = T - T_a$$

Table 6-1. PWM duty cycle results with SVPWM computation

Sector Number	q	direction = CCW	direction = CW

1	$[0,\frac{\pi}{3}]$	PWM0 = (T - dx -dy)/2
		PWM1 = (T + dx -dy)/2
		PWM2 = (T + dx + dy)/2
			PWM0 = (T + dx + dy)
			PWM1 =(T - dx - dy)
			PWM2 = (T - dx + dy)

2	$[\frac{\pi}{3},\frac{2\pi}{3}]$	PWM0 = (T - dx + dy)/2
		PWM1 =(T - dx - dy)/2
		PWM2 = (T + dx + dy)/2
			PWM0 = (T + dx + dy)/2
			PWM1 = (T + dx - dy)/2
			PWM2 =(T - dx - dy)/2

3	$[\frac{2\pi}{3},\pi]$	PWM0 = (T + dx + dy)/2
		PWM1 = (T - dx - dy)/2
		PWM2 = (T + dx - dy)/2
			PWM0 = (T - dx + dy)/2
			PWM1 = (T + dx + dy)/2
			PWM2 = (T - dx - dy)/2

4	$[\pi,\frac{4\pi}{3}]$	PWM0 = (T + dx + dy)/2
		PWM1 = (T -dx + dy)/2
		PWM2 = (T - dx + dy)/2
			PWM0 = (T - dx -dy)/2
			PWM1 = (T + dx + dy)/2
			PWM2 = (T + dx - dy)/2

5	$[\frac{4\pi}{3},\frac{5\pi}{3}]$	PWM0 = (T + dx - dy)/2
		PWM1 = (T + dx + dy)/2
		PWM2 = (T - dx - dy)/2
			PWM0 = (T - dx - dy)/2
			PWM1 = (T - dx + dy)/2
			PWM2 = (T + dx + dy)/2

6	$[\frac{5\pi}{3},2\pi]$	PWM0 = (T - dx - dy)/2
		PWM1 = (T + dx + dy)/2
		PWM2 = (T - dx + dy)/2
			PWM0 =(T + dx -dy)/2
			PWM1 = (T - dx -dy)/2
			PWM2 = (T + dx + dy)/2

The Table 6-1 shows all duty cycle values for each sector number. For a sector number the duty cycle for the six half bridge is expressed by:

UH = PWM0 and UL = T-(PWM0)

VH = PWM1 and VL = T-(PWM1)

WH = PWM2 and WL = T-(PWM2)

The two rotating directions are described. In case of CCW direction, that means the rotor follows the sequence Sector number = {6,5,4,3,2,1} and in case of CW direction, the sequence {1,2,3,4,5,6}.

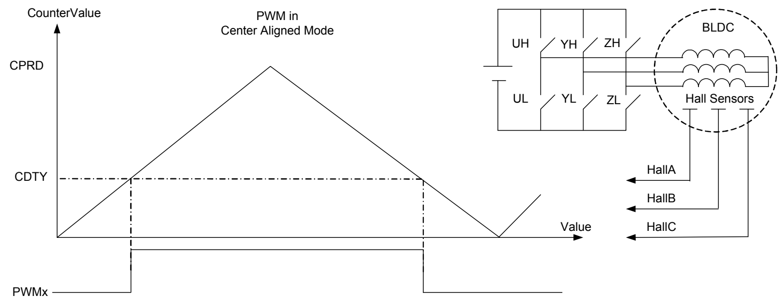
6.2.1 Pulse Width Modulation Generation

6.2.1.1 Using PWM module

The AVR32 UC3 is able to generate PWM signals through a dedicated PWM module or through the Timer Counter used. The PWM module is used in this application [(1)] .

- We need first to setup alternate functions of GPIO corresponding to alternate pins/functions used for PWM channels.
- PWM module is configured in center aligned mode or left aligned mode. We will only consider PWM module in center aligned mode to fit with S.V.P.W.M implementation. To configure this mode, we need to set:
  - CALG bifield in CMRx register at ‘1’ (fixed value)
- The PWM module has its own global prescaler.
  - PREA,PREB bitfield in PWM.MR register
- Each PWM channels has its own prescaler as well. It is possible to define a different period for a given channel **x**.
  - CPRE bitfield in PWM.CMR**x** register
- The PWM period and duty cycle is configured from this prescaled period:
  - CPRD bitfield in CMRx register
  - CDTY bitfield in CMRx register
- Finally enable PWM channel
  - CHIDx id in ENA register

Figure 6-3. PWM Channel in center aligned mode



Let’s take the following example:

Table 6-2. PWM calculation example

PBA(MHz)	CMRx.CPRE	MR.DIVA	MR.DIVB	CPRDx	PWM(Hz)
24	1	0	0	256	23437

$$PERIOD = \frac{2 \times CPRDxDIVA}{PBA}(s)$$

1. Refer to UC3 datasheet section PWM for more details

6.3 Tick Reference Generation

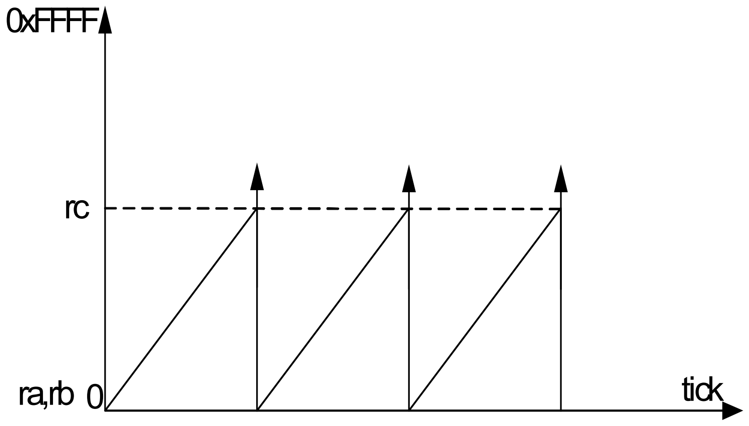
The tick reference is generated with a timer counter usage.

The AVR32 UC3 family is able to generate timer event signals through a dedicated Timer Counter module or through CPU cycle counter mode. Only the Timer Counter is used in this application [(1)] .

Timer Counter module could be used in Capture Mode or Waveforme Mode. We will only consider Timer Counter module in Waveforme mode.

In this mode, the counter will count up until a compare value (RC compare register), reset and restart from 0x0000. It is possible to generate an interrupt upon RC compare match. This interrupt occurs every tick period.

Figure 6-4. Tick generation



- The Timer Counter module has its clock selection source that the user can select:
  - TCCLKS bitfield in TC.CMR register
- RC compare register will define the periodicity of interrupt:
  - RC bitfield in TC.RC register
- To enable Timer Counter channel, set:
  - CLKEN in TC.CCR registe)

Let's take the following example:

Table 6-3. Timer Counter calculation example

PBA(MHz)	CMR.CCLKS	CMR.WAVE	CMR.WAVESEL	RC	Tick(Hz)
24	1/(FPBA/8)	1	2	4000	1000

The Tick period value is equal to

$$tick = \frac{1}{FPBA \ / 8} \times RC$$

1. Refer to UC3 datasheet section Timer Counter for more details

6.4 Hall Estimator

Each Hall sensor is connected to GPIO. All GPIOs are able to generate interrupts.

By default, interrupts are configured to be used with pin level changes.

6.4.1 Sector determination

The Sector Determination is based on the reading of the three Hall sensors. During one electrical revolution, the three Hall sensors generate 6 steps. These 6 steps divide the circle into 6 sectors which will be used in the SVPWM

6.4.2 Speed determination

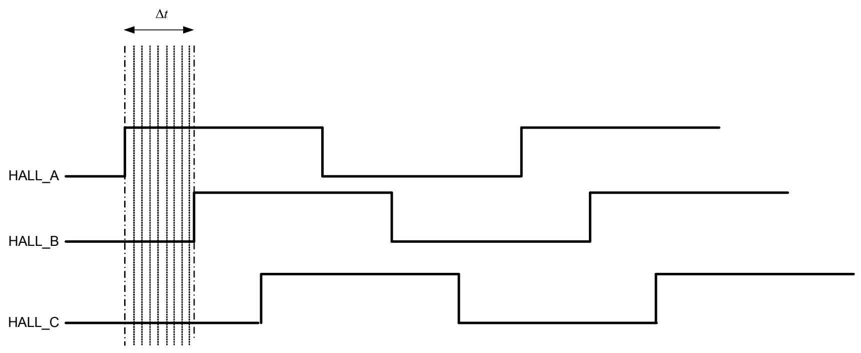
In order to determine rotor speed, we need to determine time elapsed between two interrupts.

The AVR32 UC3 microcontroller is able to measure time elapsed between two events using a dedicated timer counter. This counter computes the number of CPU cycles elpased in a certain time period. In this case, the time elapsed is equal to:

$$\Delta t(n) = n \times \frac{1}{F_{CPU}}$$

After having measured time elapsed between two interrupts.

Figure 6-5. Hall sensors signals



The electrical speed of motor is defined as

$$Speed_{electrical}(Hz) = \frac{1}{6 \times \Delta t}$$

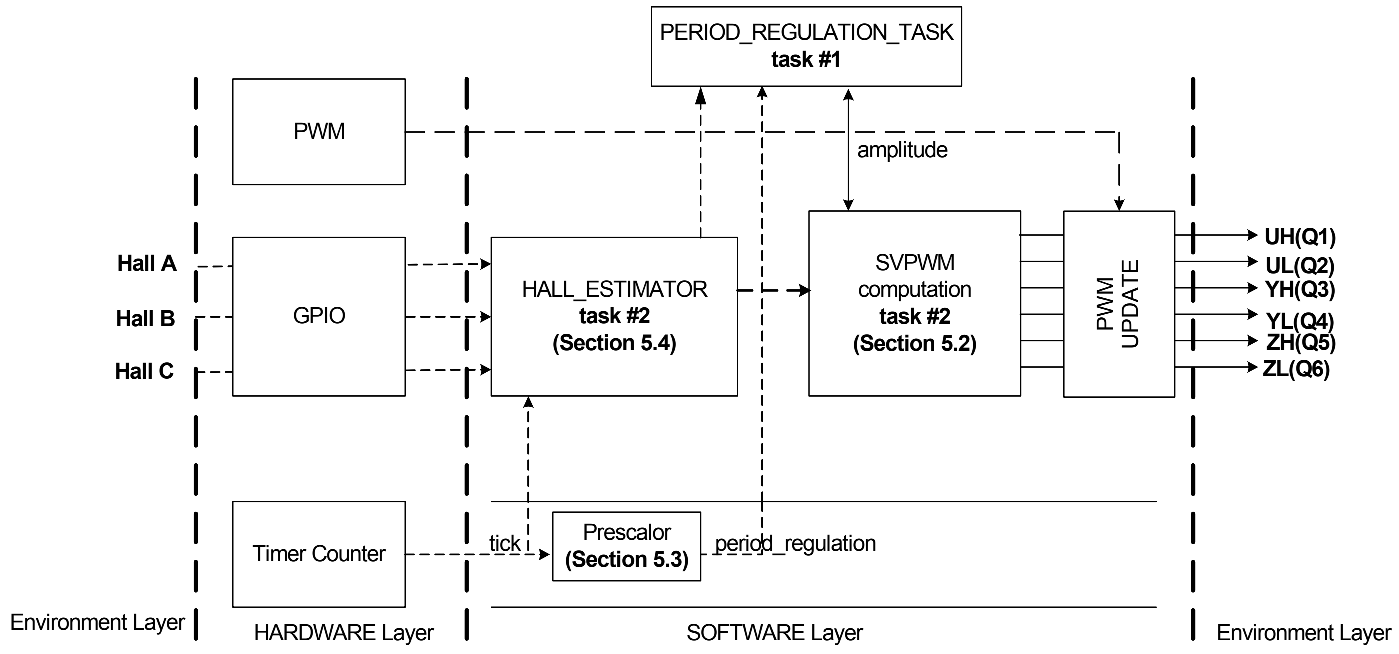
$$Speed_{mecanical}(Rpm) = \frac{60 \times Speed_{electrical}}{nbpoles}$$

For example, with n=4 and Fcpu = 48MHz

$$nbpoles = 4$$
$$Speed_{electrical}(Hz) = 89\ Hz$$
$$Speed_{mecanical}(rpm) = 1335\ rpm$$

7. Software Implementation

Figure 7-1. Block diagram



This application is based on a loop regulation.

- The timer counter generates an interrupt based on a clock reference. This reference named *tick* is prescaled to generate the *period regulation* .
- The *hall_estimator* computes the *motor position* from the GPIO inputs and the *motor speed* from *tick* reference value.
- The *period_regulation_task* computes a new value of *amplitude* from the last value of amplitude and a new *motor speed* value each *period regulation* clock.
- The SVPWM uses the *motor_position* and the *amplitude* to update all duty cycles. These new duty cycles values are used to update the PWM channel duty cycle.

Periodically, the amplitude is updated with:

$$AMPLITUDE = (K_p) \times (Speed_{reference} - Speed_{feedback}) + (K_I) \times \int (Speed_{reference} - Speed_{feedback})$$

7.1 Source Code Package Description

The software is available in the attached project on the Atmel web Site. The AVR32710.zip contains the project for the UC3A0512 Rev. E (engineering samples). The EVK110x-MOTOR-CONTROL-X.Y.Z.ES supports the UC3A0512 Rev. E only.

HTML documentation is included in the package. Use the readme.html file in the doc directory to start viewing the documentation.

- DRIVERS\

This directory contains all libraries used in the project: especially DRIVERS and DSP library support.

- SERVICES\MOTOR_CONTROL\
  - HALL_ESTIMATOR\

This directory contains the service to determine rotor position (angle and sector).

  - MOTOR_DRIVER\BLDC_MOTOR\

This directory contains the service for low level drive of PWM channels.

  - SVPWM\

This directory contains the Space Vector computation algorithm.

- APPLICATIONS\EVK110x-MOTOR-CONTROL\BLDC-SVPWM\EXAMPLE\
  - bldc_svpwm_example.c

This file contains the main() with all CPU initialization and task launch.

  - mc_control.c

This file defines the main control loop sequence.

  - mc_driver.c

This files defines the low level loop sequence access.

  - sensor_task.c

This files defines dedicated task function for display and motor control.

7.2 CPU Load & Memory Usage

The following benchmarks have been done on AVR32-GCC 4.2.2-atmel.1.0.4 with this

configuration

- FCPU = 42MHz
- FPBA = 21MHz

Table 7-1. Microcontroller utilization rate

Function	Parameters	Activation time	Activation period
COMPARE_INT_HANDLER	HALL_ESTIMATOR	5us	64us
	SVPWM_COMPUTATION	20us	64us

8. Hardware Implementation

8.1 EVK1100 Connection

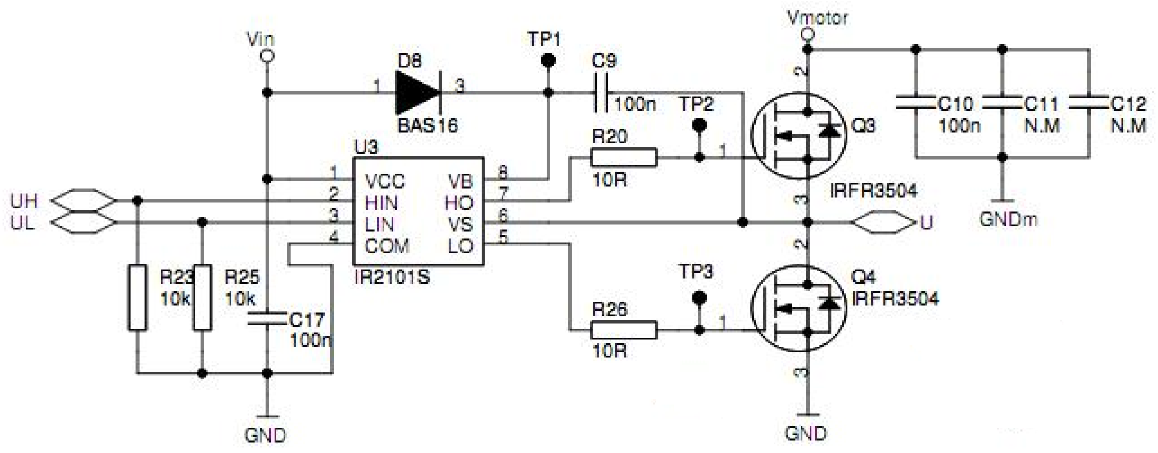
The Table 8-1 lists the connections between evaluation kit and motor connection.

Table 8-1. EVK1100 and Power Bridge Connection

EVK1100	Power Bridge
PWM0	UH
PWM1	UL
PWM2	VH
PWM3	VL
PWM4	WH
PWM5	WL
GPIO/10	H1
GPIO/29	H2
GPIO/30	H3

8.2 Power Bridge Presentation

The power bridge board is made up of standard complementary MOSFETs IRF3504.A adaptation stage is used to convert digital signals into analog signals to correctly adress MOSFETs. In this case, MOSFET drivers are used, IR2101.r



9. Running the application

9.1 Loading the Code

The AVR32 UC3 ISP solution offers an easy way to download files into AVR32 products on Atmel Evaluation Kits through the JTAG link (via the JTAGICE mkII debugger tools) or the USB bootloader.

Follow the steps below to build the application, load and run the code:

If you are using GCC with the AVR32 GNU Toolchain :

- Make sure the board is powered off.

- Plug power cable on EVK1100 and power it at 12V.

- In case you use a JTAG link:

– - Plug the JTAGICE mkII between the PC and the EVK1100 using the JTAG connector.

– - Open a Cygwin or a Linux shell, go to the *APPLICATIONS/EVK110x-MOTOR-CONTORL/AT32UC3A0512ES/GCC* directory and type :
make rebuild program run

- In case you use USB bootloader:

– - Plug the USB cable between the PC and the EVK1100 using the USB connector.

– - Open a Cygwin or a Linux shell, go to the *APPLICATIONS/EVK110x-MOTOR-CONTORL/AT32UC3A0512ES/GCC* directory and type :
make rebuild isp program run

If you are using AVR32 Studio:

– Please follow the UC3 Software Framework procedure in application note **AVR32008**

If you are using IAR Embedded Workbench for Atmel AVR32:

- Make sure the board is powered off.

- Plug the JTAGICE mkII between the PC and the EVK1100 using the JTAG connector.

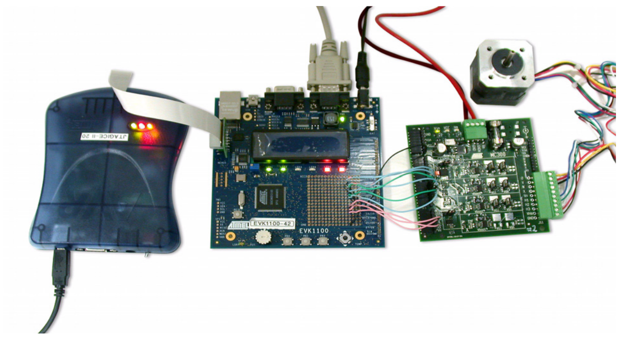
- Plug power cable on EVK1100 and power it at 12V.

- Open IAR® and load the associated IAR project of this application (located in the directory *EVK110x-MOTOR-CONTORL/AT32UC3A0512ES/IAR*).

- Press the “Debug” button at the top right of the IAR interface.
The project should compile. Then the generated binary file is downloaded to the microcontroller to finally switch on the debug mode.

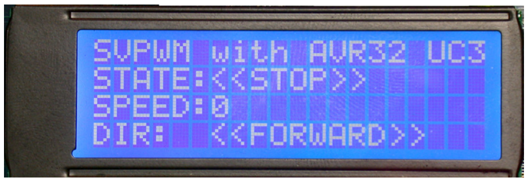
- Click on the “Go” button in the “Debug” menu or press F5.

The code then starts running.



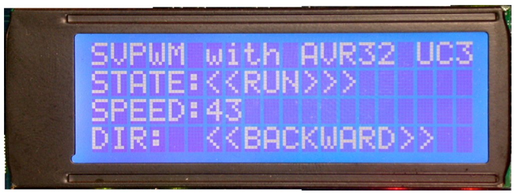
9.2 Running the Application

- Power up the board. The LCD displays :
  - State of the motor (*Run* or *Stop*)
  - Speed Motor value (In percent of maximum value)
  - Direction (*Forward* or *Backward*)



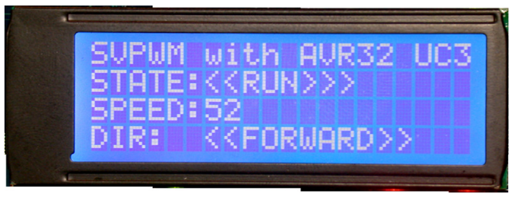
```
SVPWM with AVR32 UC3
STATE:<<STOP>>
SPEED:0
DIR:  <<FORWARD>>
```

- Push Up the Joystick. The motor starts to run



```
SVPWM with AVR32 UC3
STATE:<<RUN>>
SPEED:43
DIR:  <<BACKWARD>>
```

- Push Right the Joystick. The motor turns in forward direction.



```
SVPWM with AVR32 UC3
STATE:<<RUN>>
SPEED:52
DIR:  <<FORWARD>>
```

- Note that speed value is changed with potentiometer