Introduction

The STM8 based universal motor controller example software is written in the C programming language and provides the main functions of line synchronization, synchronized Triac gate pulse generation, tachometer interface, and speed regulation.
1 Theory of operation

The line synchronization is received at the Timer1 channel 3 input capture pin. A rising edge at TIM1-CH3 triggers a capture of the present value of TIM1 and also generates the execution of the interrupt service routine which hosts the “Zero Cross State Machine”. TIM1 is a 16-bit timer which is allowed to free run. The TIM1 input clock is prescaled so that each timer “tick” represents 0.5 microseconds. This scaling allows easy scheduling of time intervals up to 32,767 microseconds or 32.767 milliseconds. Since the longest nominal time period that a phase controller such as this would need to deal with is 20 milliseconds (the period of a 50 Hz AC line), this provides a good compromise between range and resolution. The captured value of TIM1 is stored as the global variable zctime and all gating events are scheduled relative to the latest zctime.

The enabling and disabling of Triac gating pulses are “scheduled” using the output compare functionality of TIM1. The channel 1 output compare function is used and these events trigger the interrupt that hosts the gating state machine. The output compare event does not directly trigger any hardware outputs, just the interrupt.

Aside from the input capture and the output compare interrupts, the program also employs a fixed frequency 64 microsecond “heartbeat” interrupt. This interrupt hosts the “Hallstate” state machine and the state variable speed observer. The purpose to the hallstate state machine is to poll the state of the tachometer signal and detect all rising and falling edges (with digital noise filtering). Whenever an edge is detected, the global variable position is incremented. Position gives a running indication of the rotational position of the motor starting from the time it is started. Position is cleared with each motor start. The speed observer takes position as its input and generates positionest, which tracks position. Positionest is generated within the observer by integration of another variable called speedest. So long as positionest continues to track position, speedest is a faithful estimate of motor speed. The reason for using the observer method to derive motor speed from the tachometer signal rather than the more straight forward methods of either counting tachometer edges in a fixed time interval (frequency measurement) or capturing the interval between edges (period measurement) is that the observer does an excellent job of rejecting noise regenerated by actual electrical noise in the tachometer signal and also time jitter noise caused by inconsistencies in the rotational interval between tachometer edges. Cheap tachs usually have this sort of edge jitter and it is difficult to eliminate when using an input capture technique. The direct frequency measurement technique is not very susceptible to noise but this method tends to generate a very low effective bandwidth (update rate) for the speed estimate since low pulse rate tachs must be counted over a relatively long interval in order to get sufficient resolution, especially at lower speeds.

A potentiometer provides a scaled speed command for the motor. The speed command is processed by an acceleration/deceleration control block to produce a speed reference. The speed reference is presented to a more or less conventional PI regulator which generates the gatedelay. The gatedelay is used as the command input to the zero cross and gating state machines to control Triac gate timing relative to the line zero crossing.
2  Zero cross state machine pseudocode

Case 0:
Wait 30 line cycles for system to settle, then advance to case 4

Case 4:
Zctime = captured tim1
Advance to case 10

Case 10:
Lineperiod = new captured tim1 - zctime
Lineperiodsum = lineperiodsum + lineperiod
If (16 lineperiods have been accumulated)
{ Halfperiod = lineperiodsum/32
  Usablehalfperiod = 85% of halfperiod
  Gatedelay = usablehalfperiod
  Advance to case 15
}

Case 15:
Zctime = captured tim1
Gatetime = zctime + halfperiod -500
Set gstate to 100
Schedule next gstate interrupt at gatetime
Advance to case 20

Case 20:
Zctime = captured tim1
Gatetime = zctime + gatedelay
Schedule next gstate interrupt at gatetime
Set gstate to 0
// end of zero cross state machine (state stays at case 20 until motor is re-started)
3 Gating state machine pseudocode

Note: Very first case to run is 100, which runs only once.

Case 0: // first gating pulse
If (run) take gate pins low to trigger Triac
Gateon = TRUE
Regulate = TRUE
Gatetime = gatetime + 1000 // turn off time
Schedule next gstate interrupt at gatetime
Advance to case 10

Case 10: // first turn off
If (run) take gate pins high to turn off gating
Gateon = FALSE
Gatetime = zctime + halfperiod + gatedelay // turn on time for second gate
Schedule next gstate interrupt at gatetime
Advance to case 20

Case 20: // second gate
If (run) take gate pins low to trigger Triac
Gateon = TRUE
Regulate = TRUE
Gatetime = gatetime + 1000 // turn off time
Schedule next gstate interrupt at gatetime
Advance to case 30

Case 30: // second turn off
If (run) take gate pins high to turn off gating
Gateon = FALSE
Advance case to 0 // state set for first gating, to be scheduled by zero cross state machine

Case 100: // very first gating pulse (get output high safely)
If (run) take gate pins high to turn off gating
Gateon = FALSE
Firstgate = FALSE
Gatetime = gatetime + 100
Schedule next gstate interrupt at gatetime
Advance case to 10
// end of gating state machine
4 64 microsecond heartbeat interrupt pseudocode // hallstate state machine

Case 0: // looking for new rising edge
If( tacho signal high) advance case to 2

Case 2: // looking for second consecutive high
If( tacho signal high) advance case to 4
Else back to case 0

Case 4: // looking for third consecutive high
If( tacho signal high) advance case to 10 and increment position
Else back to case 0

Case 10: // looking for new falling edge
If( tacho signal low) advance case to 12

Case 12: // looking for second consecutive low
If( tacho signal low) advance case to 14
Else back to case 10

Case 14: // looking for third consecutive low
If( tacho signal low) advance case to 0 and increment position
Else back to case 10

// end of hallstate state machine

// state variable speed observer
Unionlong0 = positionest / 4096 // scale estimate to match position
Slong0 = position - unionlong0 // calculate observer error
Speedest = slong0 // observer error used as speed estimate
Positionest = positionest + speedest // integrate speed estimate into position estimate

// catch eventual rollover of positionest (and position) and prevent
// after “adjustment” position error will not change so there will be no disturbance
If (positionest > ¼ of full numerical range)
  { positionest = positionest - ¼ of full scale
    Position = position - 524288 // half of full scale divided by 4096
  }

// end of state variable observer
5 Background loop pseudocode

If (run)
{
    If (gateon) toggle gate pins
    Else force gating pins high
}

If (regulate)
{
    Regulate = FALSE
    Read pot ADC channel and store as 255 - value ( reverse direction sense of pot)
    If (openloop ) long0 = usablehalfperiod
    Else long0 = maxrpm
    Long0 = (long0 * potvalue)/256
    Rpmcmd = long0
    Sword0 = rpmcmd - rpmref // accel/decel block error calculation
    If (sword0>acclim) sword0 = acclim // limit positive delta
    If (sword0<-declim) sword0 = -declim // limit negative delta
    Rpmref = rpmref + sword0 // integrate delta to generate output
    Rpm = speedest * 228 / tachoppr // calculate rpm
    Sword0 = rpmref - rpm // speed error calculation
    If (sword0>errorlim) sword0=errorlim // clamp speed error to prevent math problems
    If (sword0<-errorlim) sword0 = -errorlim
    Errorintegral = errorintegral + sword0 // error integration
    If (errorintegral<0) errorintegral=0 // clamp error integral
    If (errorintegral>erintlim) errorintegral = erintlim
    Slong0 = sword0 * propgain / 256
    Execute filter1, input = slong0 : output = propterm
    intterm = errorintegral * intgain / 256
    sword0 = usablehalfperiod -propterm-intterm
    If (sword0<1000) sword0=1000 // clamp gate delay
    If (sword0>usablehalfperiod) sword0 = usablehalfperiod
    If (run is FALSE) sword0 = usablehalfperiod
    If (openloop ) sword0 = usableperiod -rpmref
    Gatedelay = sword0
6 Run push-button state machine

Case 0: // looking for falling edge of pushbutton signal
If (pushbutton signal low) advance to case 2

Case 2: // confirm logic low
If (pushbutton signal low) advance to case 4
Else return to case 0

Case 4: // confirm low again
If (pushbutton signal low) advance to case 10
Else return to case 0

Case 10: // now looking for rising edge
If (pushbutton signal high) advance to case 12

Case 12: // confirm logic high
If (pushbutton signal high) advance to case 14
Else return to case 10

Case 14: // confirm logic high
If (pushbutton signal high) advance to case 20
Else return to case 10

Case 20: // full pushbutton cycle confirmed
If (run)
{
    run = FALSE
    turn LED off

    Else
    {
        Initialize all run variables
        Run = TRUE
        Turn LED on
    }

    Return to case 0 to await next pushbutton press
// end of pushbutton state machine
}

// end of "if (regulate)" section

// end of background loop
# Variable descriptions

## Table 1. Variable descriptions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openloop</td>
<td>8-bit logic flag</td>
<td>If true, pot controls gating delay instead of speed regulator command.</td>
</tr>
<tr>
<td>Run</td>
<td>8-bit logic flag</td>
<td>Flag is true when the motor is running and false otherwise.</td>
</tr>
<tr>
<td>Runstate</td>
<td>8-bit unsigned</td>
<td>Reflects the current state of the runstate state machine.</td>
</tr>
<tr>
<td>Zcstate</td>
<td>8-bit unsigned</td>
<td>Reflects the current state of the zero crossing interrupt state machine.</td>
</tr>
<tr>
<td>Cyclecounter</td>
<td>8-bit unsigned</td>
<td>Used to count AC line cycles during an initial stabilizing delay of 30 cycles and to count out 16 cycles as line period is averaged and measured.</td>
</tr>
<tr>
<td>Gstate</td>
<td>8-bit unsigned</td>
<td>Reflects current state of Triac gating interrupt.</td>
</tr>
<tr>
<td>Gateon</td>
<td>8-bit logic flag</td>
<td>Flag is true when background toggling of Triac gate is enabled and false otherwise.</td>
</tr>
<tr>
<td>Firstgate</td>
<td>8-bit logic flag</td>
<td>Flag is initialized true to prevent background loop from starting. After line synchronization has been established (and gating signal set high), flag is cleared to allow background to run.</td>
</tr>
<tr>
<td>Potvalue</td>
<td>8-bit unsigned</td>
<td>0 to 255 represents 0 to 5 V at pot wiper. Read once per half line cycle.</td>
</tr>
<tr>
<td>Regulate</td>
<td>8-bit logic flag</td>
<td>Set in gating interrupt at each gating to trigger background to run regulator routine. Flag is cleared in background when the regulator routine is run.</td>
</tr>
<tr>
<td>Hallstate</td>
<td>8-bit unsigned</td>
<td>Reflects the current state of the hallstate state machine. This state machine manages the variable “position” based on current value and history of the tacho signal.</td>
</tr>
<tr>
<td>Halfperiod</td>
<td>16-bit unsigned</td>
<td>Measured time length of an AC line half period. This value is measured once at power up based on the average value of 16 line cycles. Units are in 0.5 μs timer ticks. Nominal 16667 at 60 Hz.</td>
</tr>
<tr>
<td>Usablehalfperiod</td>
<td>16-bit unsigned</td>
<td>85% of halfperiod. Used as the practical limit for phase delay (50 or 60 Hz).</td>
</tr>
<tr>
<td>Gatedelay</td>
<td>16-bit unsigned</td>
<td>Commanded time delay (in 0.5 μs timer ticks) from zero crossing to Triac gating.</td>
</tr>
<tr>
<td>Zctime</td>
<td>16-bit unsigned</td>
<td>Most recently captured value of free running Timer1 at AC mains rising edge zero crossing.</td>
</tr>
<tr>
<td>Lineperiod</td>
<td>16-bit unsigned</td>
<td>Most recently calculated AC mains line period (in timer ticks).</td>
</tr>
<tr>
<td>Gatetime</td>
<td>16-bit unsigned</td>
<td>Most recently calculated Triac gating time referred to Timer1.</td>
</tr>
<tr>
<td>Propterm</td>
<td>16-bit signed</td>
<td>Proportional term of PI speed regulator.</td>
</tr>
<tr>
<td>Interm</td>
<td>16-bit signed</td>
<td>Integral term of PI speed regulator.</td>
</tr>
<tr>
<td>Speedest</td>
<td>16-bit signed</td>
<td>Raw speed estimate which is integrated in 64 μs interrupt to generate the position estimate “positionest”. Each count of speedest represents 15625/4096 or 3.8147 position counts per second. For a typical tacho providing 8 pulses per rotation, this is 3.8147/8 or 0.4768 revolutions per second or 28.61 RPM. A speed of 10,000 RPM (not uncommon for a universal motor) gives a speedest of 350.</td>
</tr>
<tr>
<td>Rpmcmd</td>
<td>16-bit signed</td>
<td>Scaled speed command derived from potvalue … units are revolutions per minute.</td>
</tr>
</tbody>
</table>
### Table 1. Variable descriptions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rpmref</td>
<td>16-bit signed</td>
<td>Output of accel/decel control block. Follows “rpmcmd” but slews to respect accel/decel limits.</td>
</tr>
<tr>
<td>Rpm</td>
<td>16-bit signed</td>
<td>Current measured motor speed expressed in revolutions per minute. Feedback for regulator.</td>
</tr>
<tr>
<td>Lineperiodsum</td>
<td>32-bit unsigned</td>
<td>Sum of first 16 measured line periods used to calculate average. Unit is Timer1 ticks.</td>
</tr>
<tr>
<td>Position</td>
<td>32-bit unsigned</td>
<td>Initialized to zero, this variable accumulates all tacho pulses (rising and falling edges). This variable is adjusted whenever it crosses half of full scale (by subtracting half of full scale) so that it does not ever overflow. This variable is the input to the state variable speed observer, which calculates estimated speed.</td>
</tr>
<tr>
<td>Positionest</td>
<td>32-bit unsigned</td>
<td>The speed estimating state variable observer constantly compares this variable with variable “position” and generates an error term which is ultimately used as the speed estimate. This speed estimate is integrated in the 64 μs interrupt to generate positionest. positionest is scaled to be 4096 times bigger than “position” ... This is done in order to increase the resolution of the speed estimate variable.</td>
</tr>
<tr>
<td>Filter1int</td>
<td>32-bit signed</td>
<td>This is the integral term associated with filter1. Filter1 is a low pass filter used to smooth the proportional term of the PI speed regulator ... the regulator is thus a modified proportional plus integral algorithm.</td>
</tr>
<tr>
<td>Errorintegral</td>
<td>32-bit signed</td>
<td>This is the raw time integral of speed error used by the PI speed regulator ... units are RPM-ticks where the time tick is a half line cycle.</td>
</tr>
</tbody>
</table>
8 Compile time settings (#defines)

Table 2. Compile time settings (#defines)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maxrpm</td>
<td>numeric</td>
<td>Full scale RPM setting.</td>
</tr>
<tr>
<td>Acclim</td>
<td>numeric</td>
<td>Acceleration limit. Units are RPM per half line cycle.</td>
</tr>
<tr>
<td>Declim</td>
<td>numeric</td>
<td>Deceleration limit. Units are RPM per half line cycle.</td>
</tr>
<tr>
<td>Propgain</td>
<td>numeric</td>
<td>Proportional gain for PI speed regulator. Scaling is such that a value of 256 will imply a gating time advance of 0.5 microseconds per RPM of speed error.</td>
</tr>
<tr>
<td>Intgain</td>
<td>numeric</td>
<td>Integral gain for PI speed regulator. Scaling is such that a value of 256 will imply a gating time advance of 0.5 microseconds per RPM of speed error per half line cycle.</td>
</tr>
<tr>
<td>Filter1tc</td>
<td>numeric</td>
<td>Controls the time constant of the low pass filter which is cascaded with the proportional path of the PI speed regulator.</td>
</tr>
<tr>
<td>Tachoppr</td>
<td>numeric</td>
<td>The total number of edges (rising and falling) presented by the tachometer in each rotation of the motor. This is used to calculate RPM.</td>
</tr>
</tbody>
</table>

9 Revision history

Table 3. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-Jan-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale. Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS “AUTOMOTIVE GRADE” MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied. The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies
Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com

12/12 Doc ID 024038 Rev 1