AN1083
APPLICATION NOTE

ST72141 BLDC MOTOR CONTROL SOFTWARE AND FLOWCHART EXAMPLE

by Microcontroller Division Applications

INTRODUCTION

The software examples described in this application note are those generated by the ST7MTC1 Kanda kit. 80% of the code is generic, the remaining 20% is specific to the implementation of the ST72141 in the ST7MTC1 kit (user interface and communication).

The purpose of this application note is to give the flowcharts of the software examples showing how to drive the motor in both current and voltage modes and give examples of open loop or closed loop speed regulation.

The software examples giventhe in file attached with this Application note illustrate Current mode and closed loop driving mode for a 4-pole BLDC motor.

1 DESCRIPTION OF SOURCE FILES

The source files described in this section refer the software the file attached with this application note.

– A0ut_cc.asm is the main file and also contains the interrupt routines.

– Sub_cc.asm contains the software subroutines.

– Sub_cc.inc contains the subroutine declarations.

– Cst_cc.asm and Cst_cc.inc contains the constant declarations.

– ST72e141.asm and st72e141.inc are the map files.

Paramcc.txt is a complete set of motor parameters. These parameters are declared in the ROM part of the ST72141 (with an include file) and then transfered to the RAM part of the ST72141 at the beginning of the main file. This is because the value of some parameters can be changed while the motor is running. Most of these parameters are coded as indicated in the datasheet.

Deframp.txt is the ramp table for the starting sequence of the motor. It contains 38 consecutive decreasing step times in order to accelerate the motor and to detect the first Zero-crossing event.

The Start/Stop order of the motor is given by pin PB0.

This software (except the declaration of the motor parameters) is the one given for current mode, closed loop driving mode in the third stage of the ST7MTC1 motor control demokit.

Table of Contents

INTRODUCTION 1

1 DESCRIPTION OF SOURCE FILES 1

2 PERIPHERAL INITIALIZATION, STARTING SEQUENCE 5

3 MAIN PROGRAM 9

4 THE THREE PRINCIPAL INTERRUPT ROUTINES (C, D AND Z) 11

4.1 FIRST PART OF C AND D INTERRUPT ROUTINES 11

4.2 BODY OF C INTERRUPT ROUTINE 12

4.3 SYNCHRONOUS MODE PART OF C INTERRUPT ROUTINE 14

4.4 FIRST PART OF Z AND R INTERRUPT ROUTINES 16

4.5 BODY OF Z INTERRUPT ROUTINE 17

4.6 TRANSITION FROM SYNCHRONOUS TO AUTO-SWITCHED MODE (C INTERRUPT) 18

4.7 AUTO-SWITCHED MODE PART OF C INTERRUPT ROUTINE 19

4.8 LAST PART OF C INTERRUPT ROUTINE 20

5 OTHER FEATURES 23

5.1 R INTERRUPT (MTIM TIMER RATIO CHANGE) 23

5.2 SOFTWARE DEMAGNETIZATION 25

5.3 O (MULTIPLIER OVERFLOW) AND E (EMERGENCY STOP) INTERRUPT ROUTINES 29

6 SUMMARY 30

Figure 1. BLDC control strategy

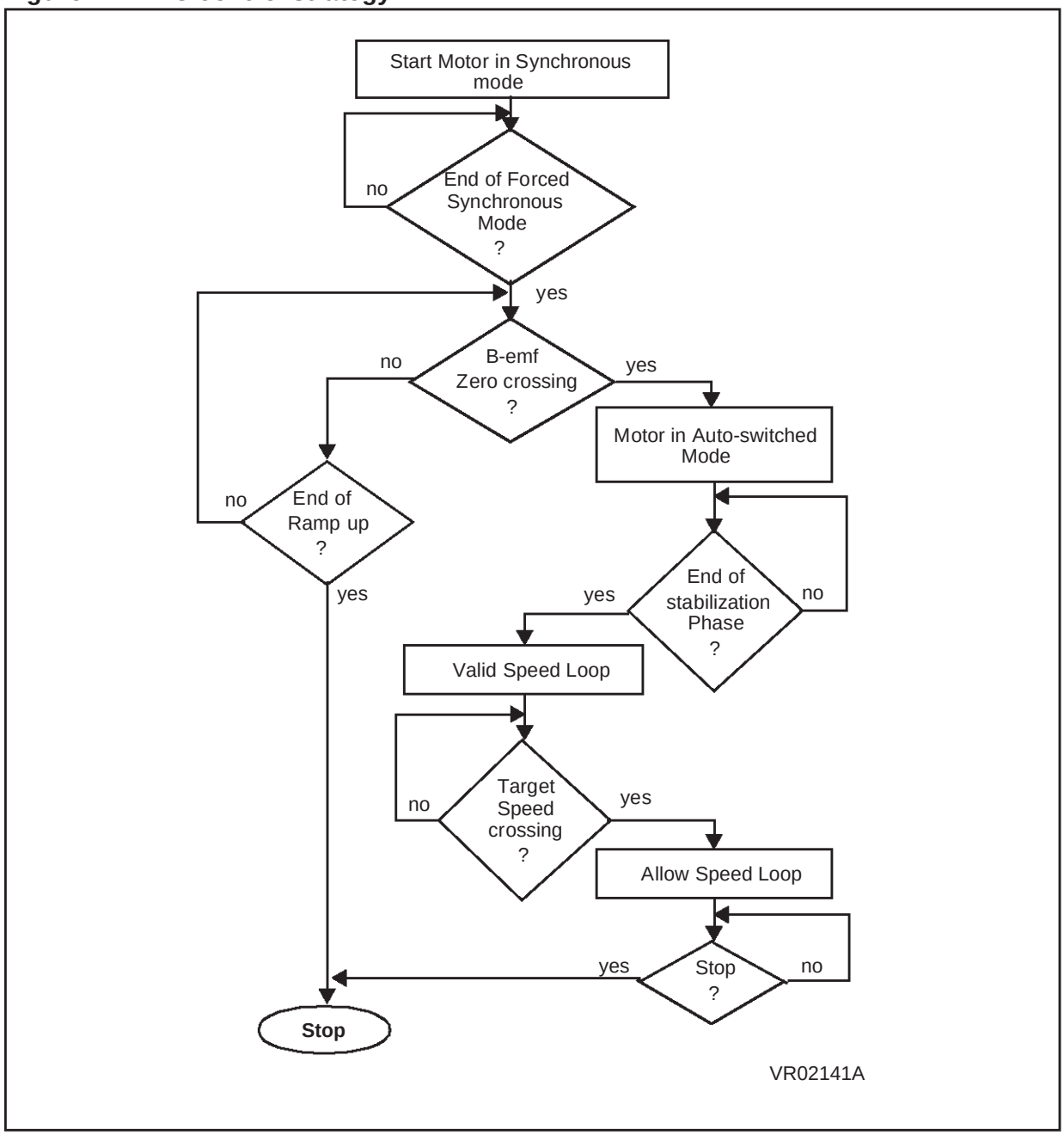


Figure 1 gives a general overview of the control software after receiving an order to start the motor.

After the start order, the motor begins to run in **forced synchronous mode**. Forced synchronous mode is part of **synchronous (switched)** mode. It is the motor starting sequence. During synchronous (switched) mode, decreasing step times are imposed in order to accel-

erate the motor and to detect the first zero crossing event (Z event), the current is imposed as well. This succession of decreasing step times is referred to as the motor ramp-up.

Forced synchronous mode corresponds to the first few steps of the ramp-up before the Z event interrupt is enabled. The number of steps in forced synchronous mode (with Z interrupts disabled) is chosen by the software programmer.

After these few steps in forced synchronous mode, we are still in the ramp-up, starting phase of the motor, and we are in synchronous (switched) mode. The step time and current are still imposed. We are waiting for a Z interrupt.

If no Z interrupt occurs before the end of the ramp-up, the motor is stopped.

When we detect the first Z event in synchronous (switched) mode, we can choose to have 2 or 3 or more consecutive Z events before putting the motor in auto-switched running mode.

Once the motor is in auto-switched mode, we wait for the end of the stabilization phase. For example, in closed loop, the stabilization phase consists of a few steps between the first step in auto-switched mode and entering the speed regulation loop. The number of steps in the stabilization phase is chosen by the software programmer.

To summarize, after the start order, the motor starts in forced synchronous mode, then enters synchronous (switched) mode when the Z event interrupt is enabled and then enters auto-switched mode.

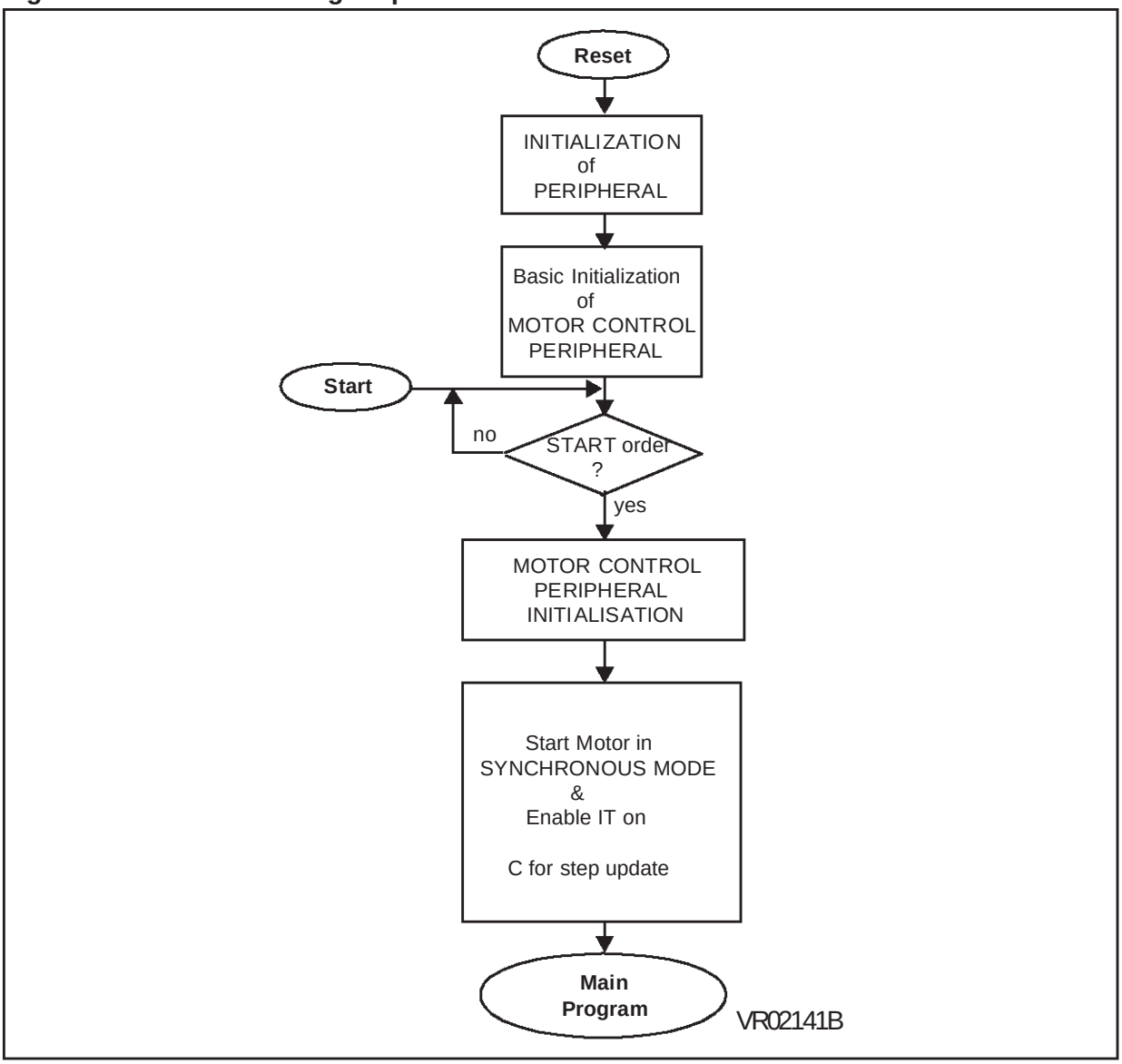
In this application note,the flowchart illustrates the starting sequence of the motor and the transition between synchronous and auto-switched modes. So, at first, the three main interrupt routines are described (C, D, Z events), then the other features are described in Section 5 of this document.

Other special cases are presented together at the end of the application note.

Note: All the names of the software routines presented in this document are strictly the same in the code generated in the ST7MTC1 kit.

2 PERIPHERAL INITIALIZATION, STARTING SEQUENCE

Figure 2. ST72141 Starting sequence flowchart



Before starting the motor, a basic initialization of the peripheral is done by software (Figure 3) and then, we wait for the start order.

First thing after the start order, we initialize the Motor Control peripheral (Figure 4) with the driving mode and the motor parameters, the motor is then started in synchronous (switched) mode and the software goes to the main program (Figure 6).

In **forced synchronous mode**, only the C event interrupt is enabled, then in **synchronous (switched) mode**, we enable the Z event interrupt as well. The D event interrupt is disabled in both **forced synchronous** and **synchronous (switched) modes**.

Figure 3. Basic initialization of Motor Control Peripheral

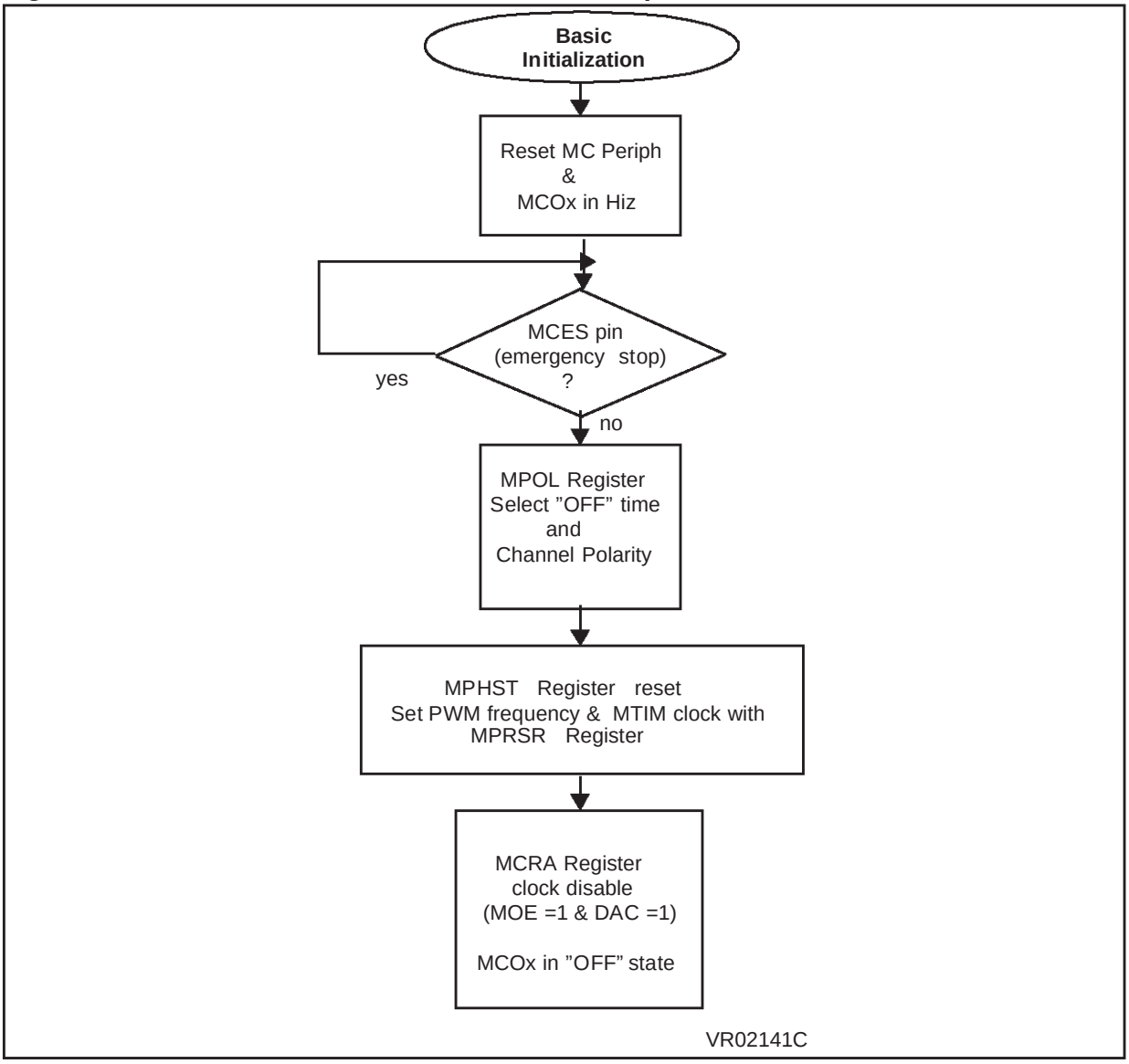


Figure 3 shows the basic initialization of peripheral by the software before the loop waiting for the order to start the motor.

This basic initialization consists of:

– Resetting the Motor Control Peripheral (by setting a control bit in the MCRA register)

– Putting all the outputs in high impedance and in off state

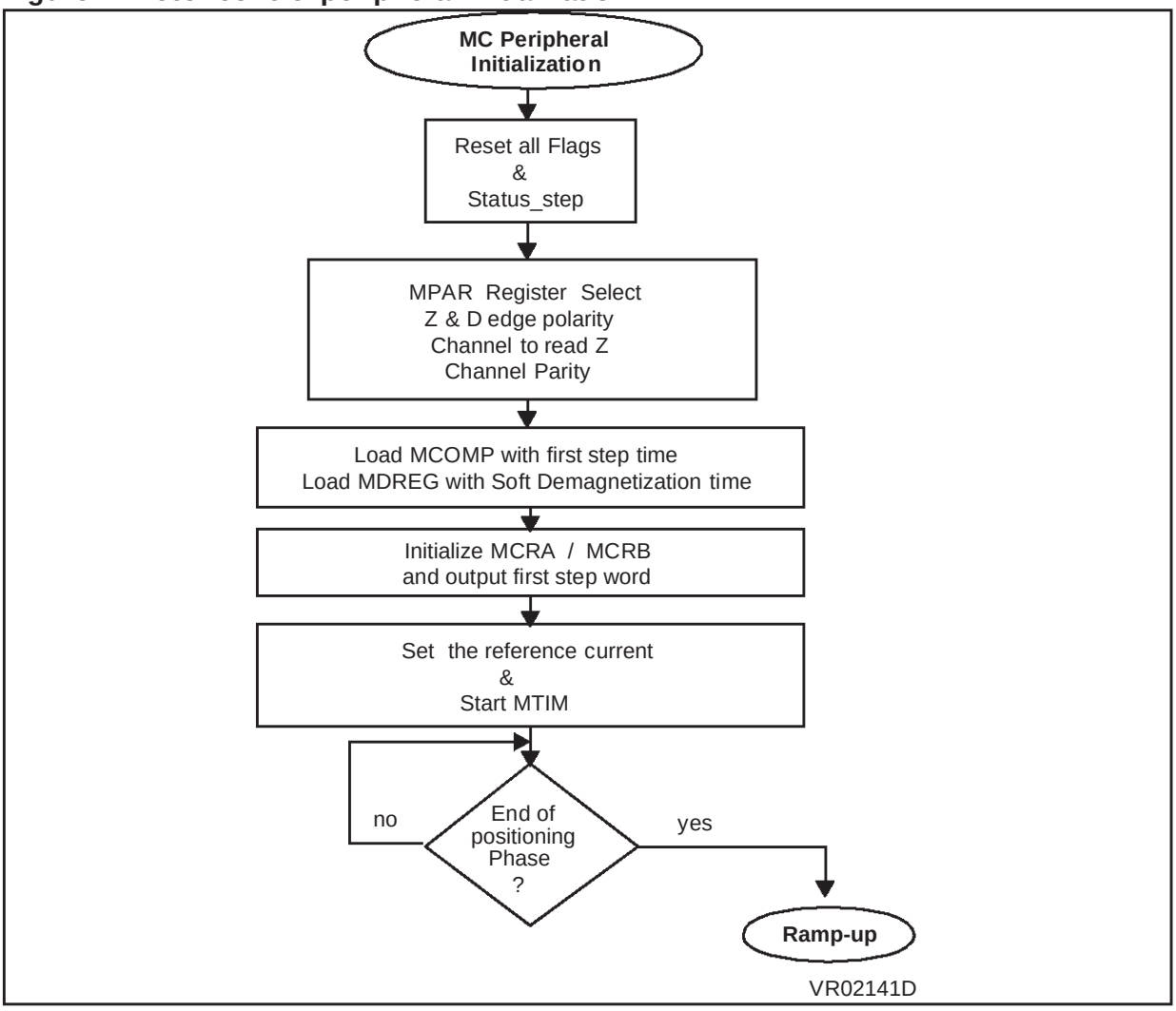
– Selecting the channel polarity (with the MPOL register)

– Selecting the internal clock frequency (current mode) and the MTIM internal 8-bit timer ratio with the MPRSR register

– Disabling the clock

Once the basic initialization is done, we wait for the order to start the motor. Then the motor control peripheral initialization is done (initialization of the driving mode and motor parameters).

Figure 4. Motor control peripheral initialization



All the flags present in the RAM variables are reset, as well as the status_step variable (RAM variable used to indicate the driving mode we are in).

We just after we receive the start order, the motor is started in synchronous (switched) mode, the step time is imposed by the ramp table in the software (the ramp table is the succession of decreasing step times that accelerate the motor during the start-up sequence). So, the first step time is loaded in the MCOMP register (in synchronous mode, the MCOMP register contains the step time)

The MDREG register is loaded with the software Demagnetization time. In synchronous (switched) mode, all the D events are simulated by software. There is no hardware end of Demagnetization event in synchronous (switched) mode.

The Z event and D event edge polarities are selected with the ZVD and CPB bits.

The MCRA and MCRB registers are initialized with the driving mode (current or voltage mode with or without sensors) and all the motor parameters are downloaded in the software.

Then follows the positioning phase. As we do not know the rotor position at first, we output a first step to position the rotor (we do it twice).

The reference current or voltage is set and the internal 8-bit timer is started by enabling the clock. Then once the positioning phase is finished, the software goes through the ramp table (ramp-up phase Figure 5) with decreasing imposed step times to accelerate the motor.

Figure 5. Synchronous starting ramp-up

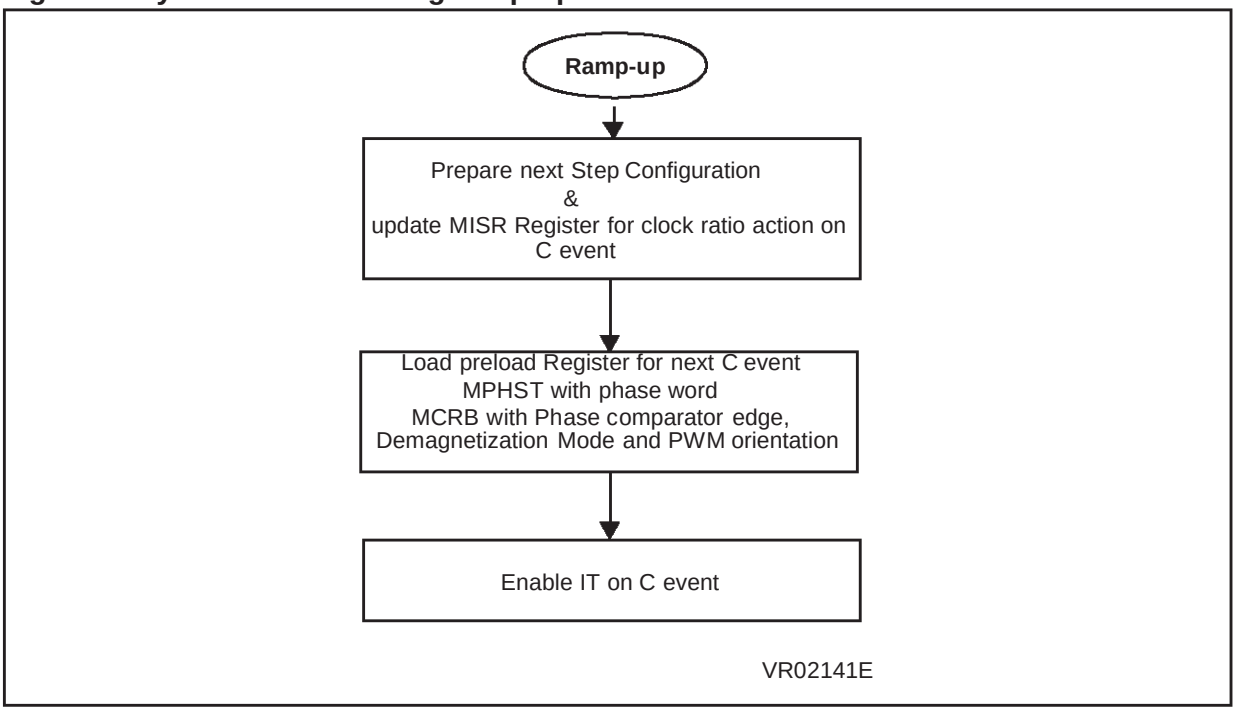


Figure 5 shows what the software does in each step during the ramp up. As the MPHST register and MCRB register have preload registers (see Application Note 1082), in each step, the data for the next step can be prepared and entered in the preload register, this data will be taken in account and loaded in the active registers at the C event.

In forced synchronous mode, only the C event interrupt is enabled. After a few steps in the ramp table, the Z event interrupt will be enabled.

3 MAIN PROGRAM

Figure 6. Main program

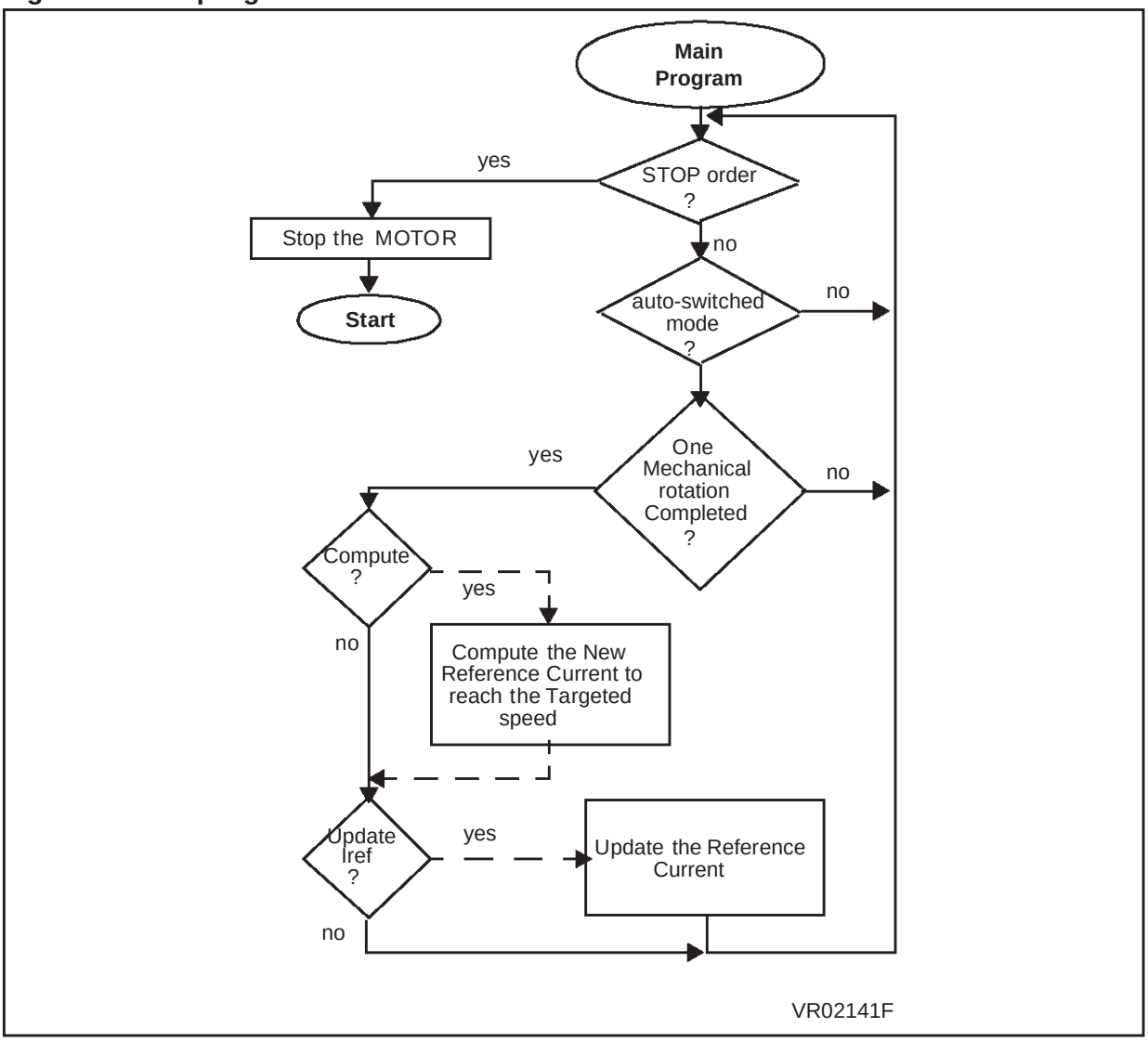


Figure 6 shows what is done in the main program.

The stop order is checked continuously. This is the only thing that is done until the motor enters the auto-switched mode. If the motor is in synchronous (switched) mode, the main program will be interrupted by a C event and a Z event. A D event is not an interrupt in synchronous (switched) mode.

If the motor is in auto-switched mode, the software waits until one mechanical rotation is completed (this is the stabilization phase, the number of steps of the stabilization phase is decided by the software programmer). In auto-switched mode, the main program can be interrupted by

all the events enabled by the software programmer. This will be described in Section 5.2 of this document.

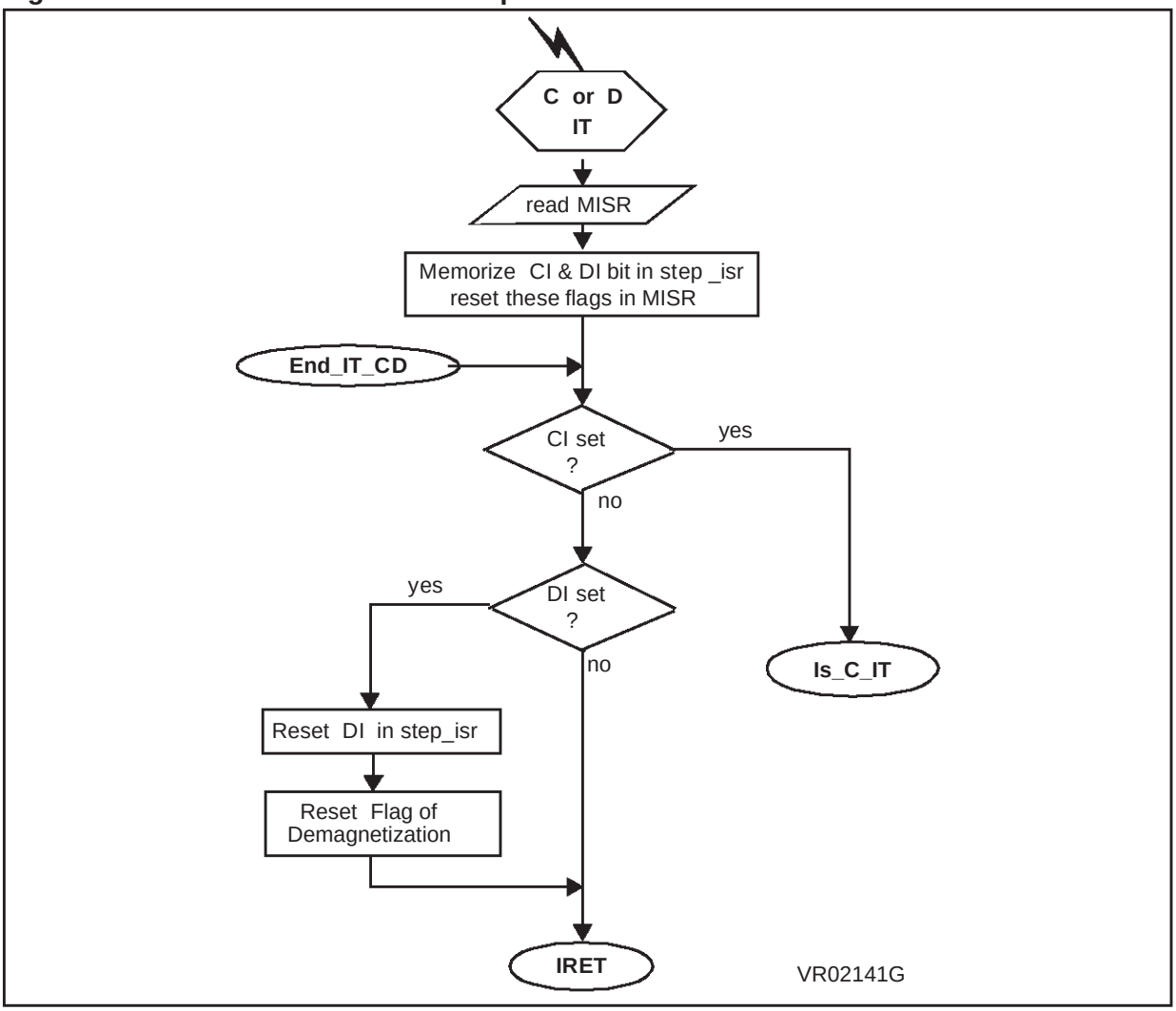
The dotted line indicates what is done in the main program in closed loop speed regulation. After the stabilization phase, the main program checks if there is a need to compute and to update a new reference current or voltage to keep the motor running at the target speed fixed by the user.

If we are still in synchronous (switched) mode, the software waits for a C event interrupt (Figure 7).

4 THE THREE PRINCIPAL INTERRUPT ROUTINES (C, D AND Z)

4.1 FIRST PART OF C AND D INTERRUPT ROUTINES

Figure 7. C event and D event interrupt routines

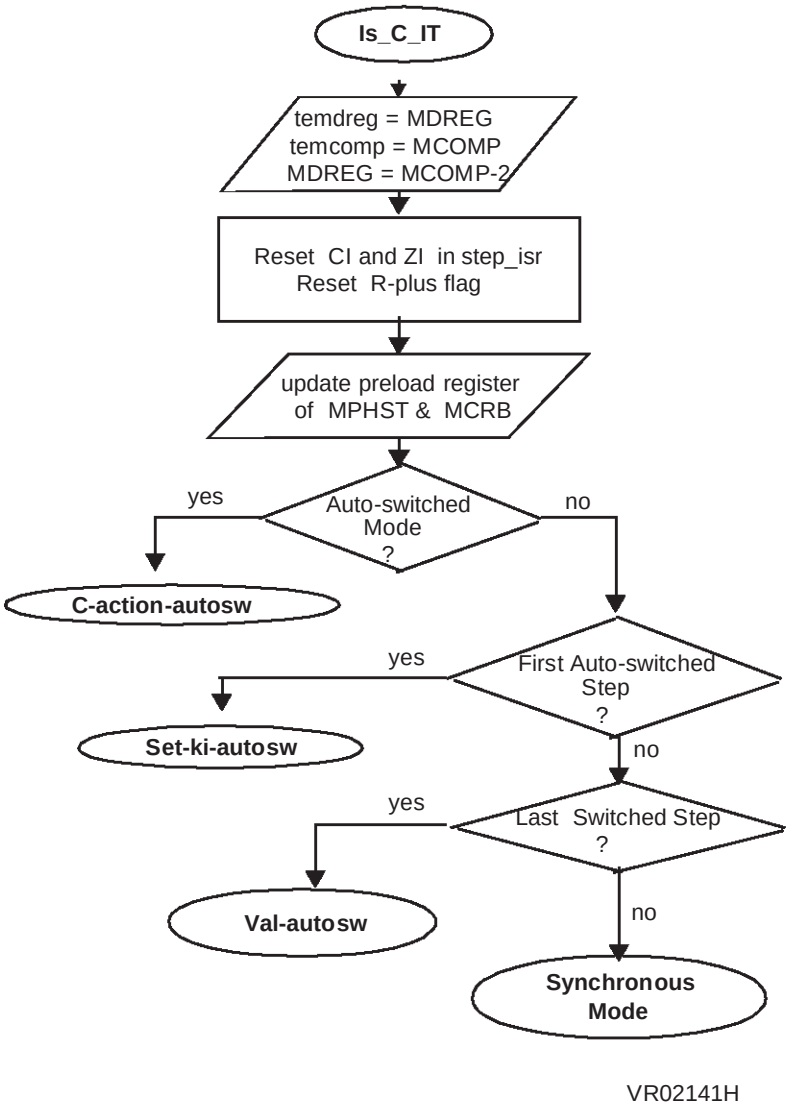


When a C event or a D event occurs, the software reads the MISR register and memorizes the interrupt flags in a RAM variable (step_isr) then these flags are reset in the MISR register.

Then the RAM variable is read in order to see what flags have been set. If the commutation flag is set (CI flag), the software enters the C interrupt routine (Is_C_IT Figure 8). If this flag is not set then the software checks if the end of Demagnetization flag is set (DI flag). If the DI flag is set, we enter the End of Demagnetization event interrupt routine. In this interrupt routine, the only thing that is done is the reset of the flag. Then the software goes back to the main program and waits for an interrupt. Figure 8 shows what is done when the commutation event interrupt flag (CI) is set.

4.2 BODY OF C INTERRUPT ROUTINE

Figure 8. Overview of the C event interrupt routine



Whatever the motor driving mode or the sequence, during a C event interrupt routine, the Demagnetization time stored in the MDREG register is memorized in a RAM variable. The value of the MCOMP register is memorized as well. Then, to avoid a parasitic Demagnetization when the interrupt routine is executed, a value less than the value in the MCOMP register is loaded in the MDREG register (meaning that the internal timer has already at that time a value greater than the value loaded in the MDREG register, so no Demagnetization interrupt will occur while this part of the interrupt routine is executed).

The interrupt flags are reset and the preload register of MCRB and MPHST are updated to prepare the data for the next C event.

Then the C event interrupt routine is divided to handle several cases:

– The motor is still in synchronous (switched) mode when the interrupt occurs

– The motor is on the last step in synchronous (switched) mode when the interrupt occurs

– The motor is in the first step in auto-switched mode when the interrupt occurs

– The motor is already in auto-switched mode when the interrupt occurs

The fact that the motor is in its last step in synchronous (switched) mode is memorized during a Z event interrupt routine (Figure 11) once the target number of consecutive Z event interrupts in synchronous (switched) mode, set by the software programmer, has been reached.

When the next C event occurs, the software will memorize that the motor is in the first step in auto-switched mode.

Let's take each case independently:

First case, the motor is in synchronous (switched) mode when a C event occurs, the software enters the C event interrupt routine and goes to the synchronous (switched) mode routine (Figure 9).

Second case, the motor is in its last step in synchronous (switched) mode when the C event interrupt occurs (this has been memorized during the previous Z interrupt routine), this time the software goes to the val_autosw routine(Figure 12). In this routine, the software memorizes that the motor is in its first step in auto-switched mode.

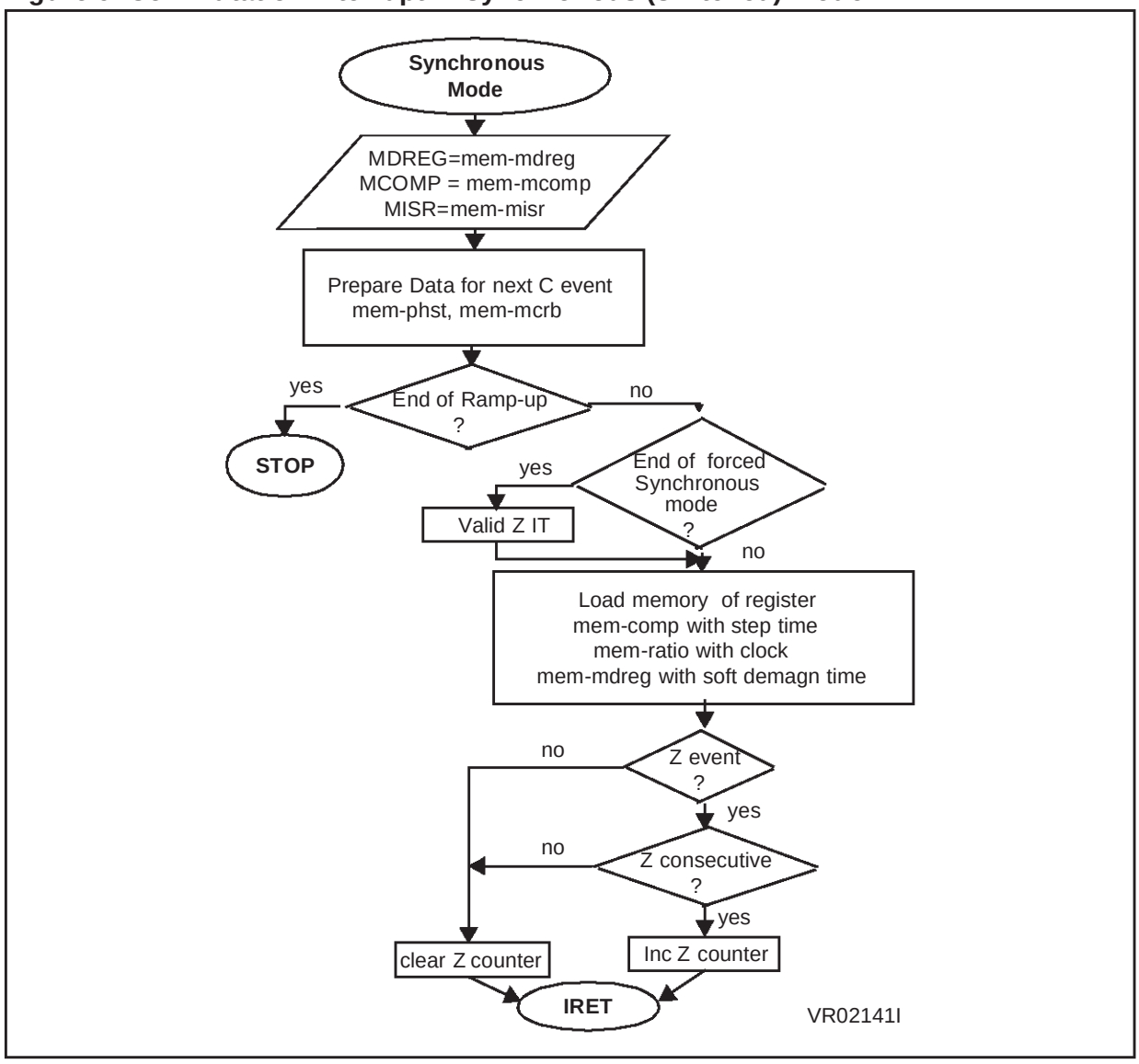
Third case, the motor is in its first step in auto-switched mode, the software goes to the Set_ki_autosw routine (Figure 13)

Last case, the motor is in auto-switched mode, the software goes to the C_action_autosw routine (Figure 13)

4.3 SYNCHRONOUS MODE PART OF C INTERRUPT ROUTINE

Let's assume that we are in the first case, the motor is still in synchronous (switched) mode when the C event interrupt occurs. So, the software goes to the synchronous mode routine (Figure 9).

Figure 9. Commutation interrupt in synchronous (switched) mode



When a commutation event occurs, the motor is in synchronous (switched) mode, so the software enters the synchronous routine in the C event interrupt routine.

The registers MDREG, MCOMP and MISR are updated for this step. The data for the next C event are prepared and memorized in RAM variables. The data will be loaded in the preload register one step before they are used.

The software checks if the motor is at the end of the ramp-up, if yes we have to stop the motor because it is not yet in auto-switched mode. If the motor is not in auto-switched mode at the end of the ramp table, it is stopped. Normally, the motor enters auto-switched mode after only a few steps.

If the motor is not at the end of the ramp table, the software checks if it is at the end of forced synchronous mode. If yes, the Z event interrupt is enabled.

All the register values are memorized by the software and it checks if a Z event has occurred in the previous step. When a Z event interrupt occurs, a counter is incremented until the number of consecutive Z events reaches the target number set by the software programmer and enters auto-switched mode.

Then the software goes back to the main program (Figure 6) and waits for the next interrupt.

If we assume that we are in synchronous (switched) mode and that the Z interrupt has been enabled, the next interrupt in the main program could be a Z event and the software will go through the Z event interrupt routine (Figure 10).

Note that we are assuming for the moment that we are in synchronous (switched) mode so only the C and Z event interrupts have been enabled by software.

4.4 FIRST PART OF Z AND R INTERRUPT ROUTINES

Figure 10. R (ratio) and Z (BEMF) interrupt routine

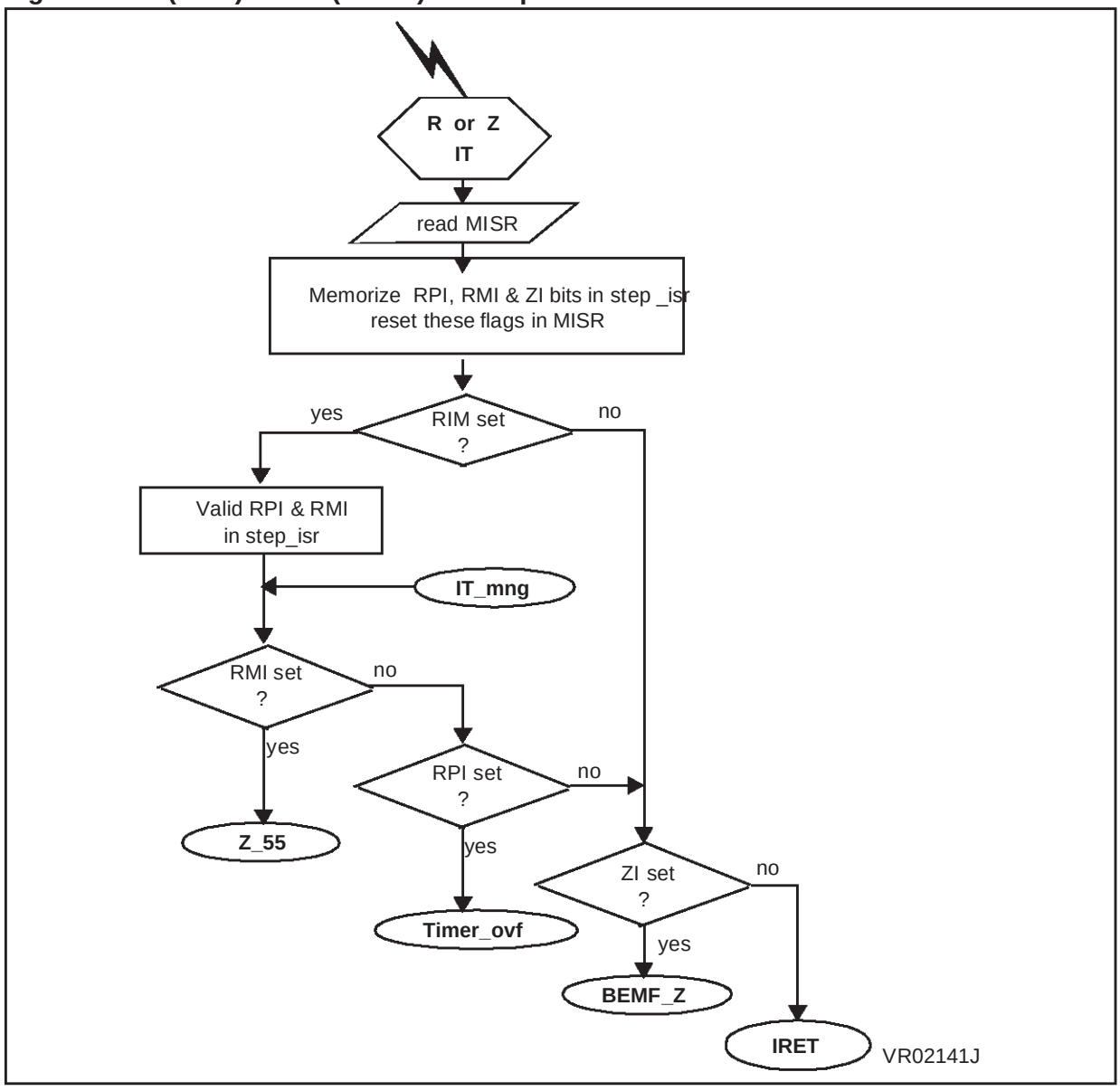
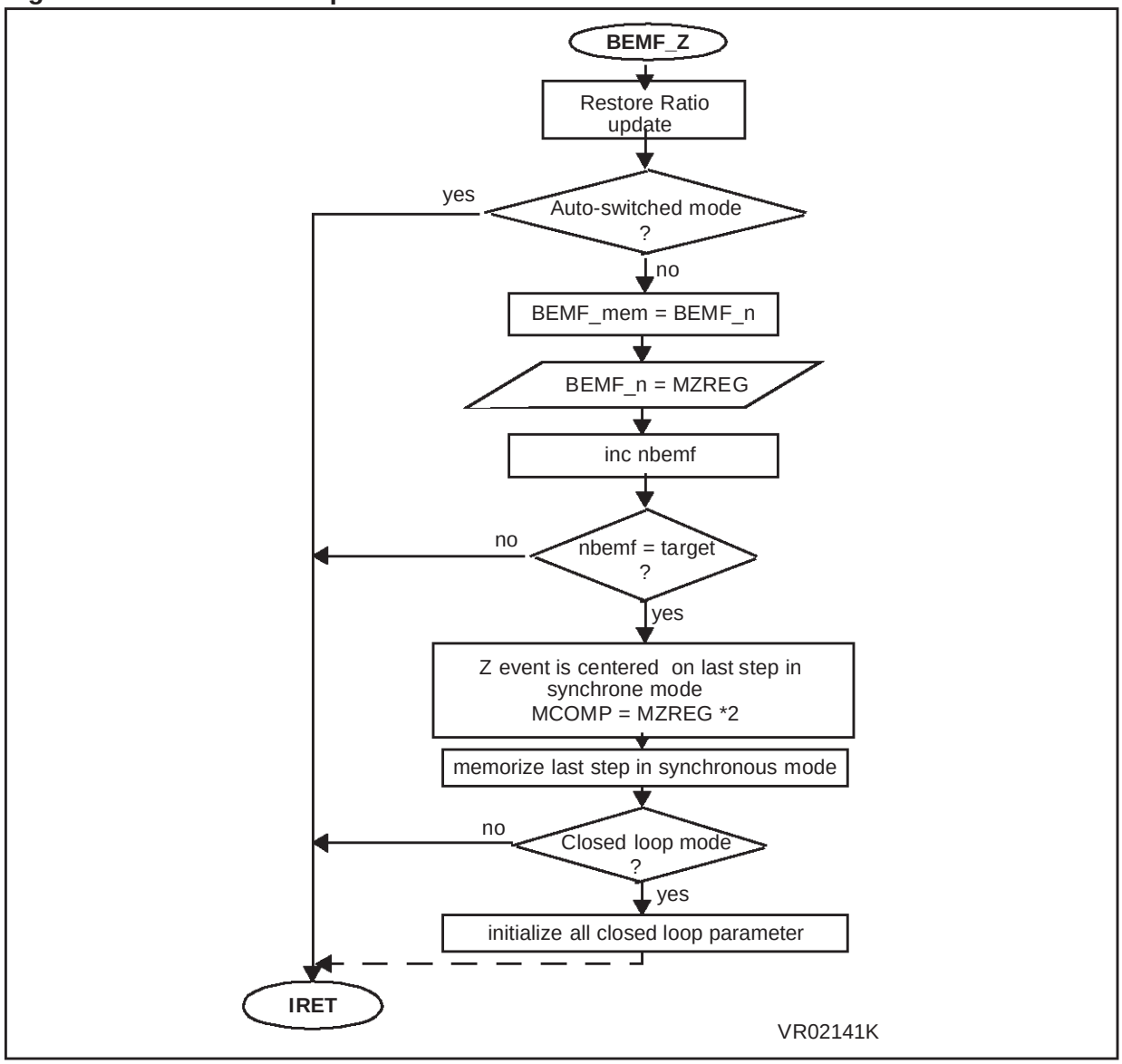


Figure 10 shows the beginning of the R (ratio) and Z (BEMF zero crossing) interrupt routines.

These routines read the interrupt flags, memorize them in a RAM variable and then reset the flags. If the R event (ratio) flag has been set, the software enters the R event interrupt routine depending on whether it is a R- or a R+ event. This will be explained in the special cases at the end of this application note (Figure 16 and Figure 17). If the Z event flag (ZI) has been set in the MISR register, the software goes through the Z event interrupt routine BEMF_Z (Figure 11). If not, the software goes back to the main program and waits for the next interrupt.

4.5 BODY OF Z INTERRUPT ROUTINE

Figure 11. Z event interrupt routine



In auto-switched mode, nothing is done in the Z event interrupt routine. An interrupt is generated and the software just goes back to the main program (Figure 6). If the motor is still in synchronous (switched) mode then the zero crossing times (current and previous) are memorized in two RAM variables. Then the counter of consecutive Z events is incremented. If the number of consecutive zero-crossing events is equal to the target number fixed by the software programmer, the software memorizes that the motor is in its last step in synchronous (switched) mode.

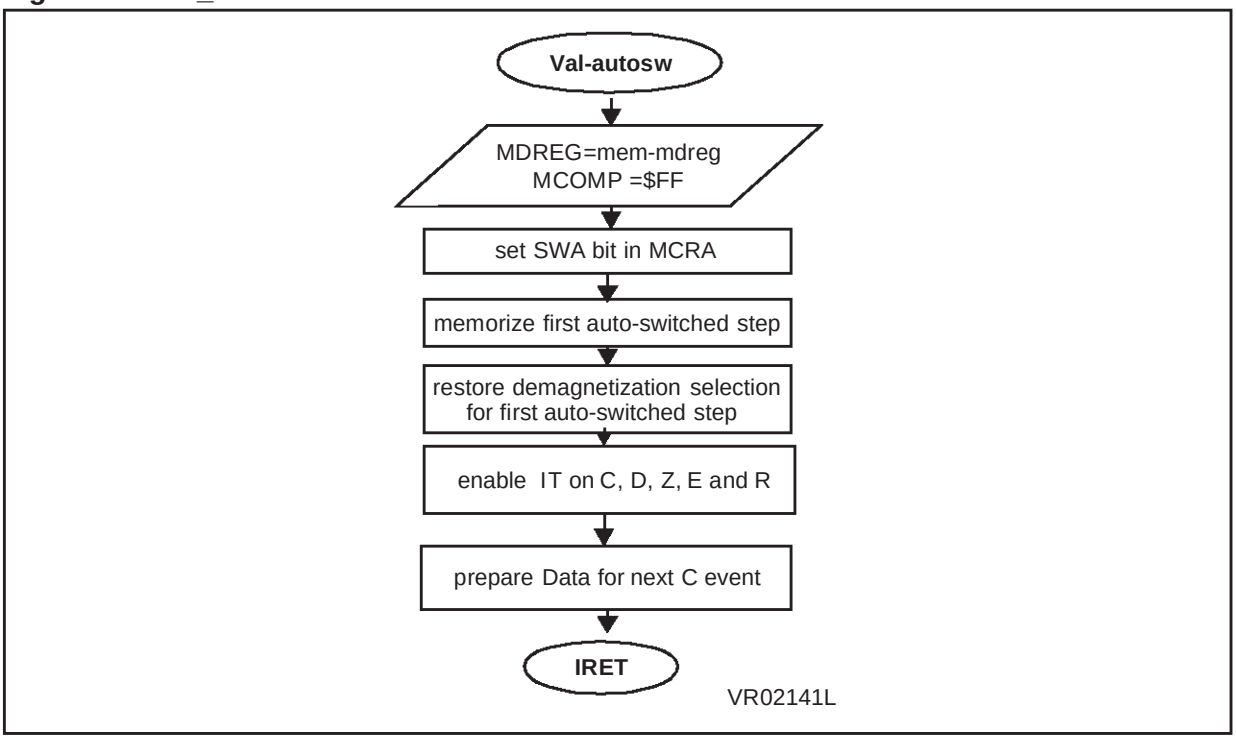
The Z event time is centered on the current step, meaning that the value entered in the MCOMP register (step time in synchronous mode) is twice the value in the MZREG register.

If the motor is in closed loop mode (speed regulation), then the initialization of the closed loop parameters is done at this point in the program.

The software then goes back to the main program (Figure 6) and waits for the next interrupt which will be a C event. As we can see in Figure 8, if the software has memorized that the motor is in its last step in synchronous (switched) mode, we will enter the Val_autosw routine (Figure 12) at the next C event, as described below.

4.6 TRANSITION FROM SYNCHRONOUS TO AUTO-SWITCHED MODE (C INTERRUPT)

Figure 12. Val_autosw flowchart routine



Let's assume we are still in synchronous (switched) mode, the last step in that mode. The software has entered the corresponding C event interrupt routine (see Figure 8).

In this routine, auto-switched mode is enabled. So, at first, the maximum value is loaded in the MCOMP register to avoid a parasitic C event during the enabling routine.

The SWA bit in the MCRA register is set to enable auto-switched mode. Now the internal 8-bit timer MTIM will be reset on a Z event.

The software memorizes that the motor is in its first step in auto-switched mode.

The demagnetization selection for the first auto-switched step is restored and as the motor enters auto-switched mode, all the interrupts are enabled.

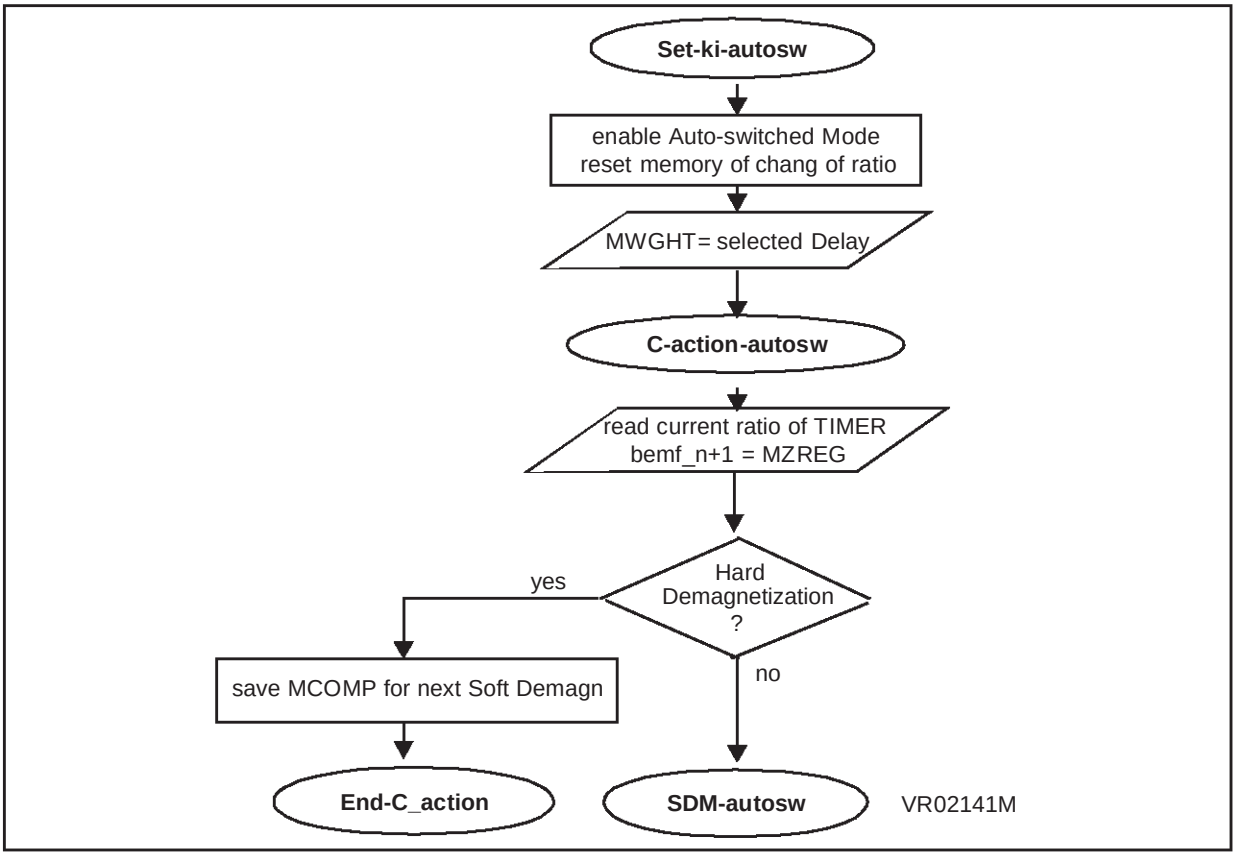
The E event and R event interrupts will be explained and described in Section 5 of this document (see Figure 16, Figure 17 and Figure 21).

The data for the next commutation event are prepared and memorized to be loaded in the preload registers.

Then the software goes back to the main program and waits for the next interrupt which will be a D event. But as we have seen in Figure 7 nothing is done by the interrupt routine except to store the DI bit, clear the MISCR flags then return to the main program and wait for the next interrupt. Then the next interrupt will be a Z event, but as the motor is now in auto-switched mode (first step in this mode) nothing is done by this interrupt routine either (as we can see in Figure 11). The software will again go back to the main program, and wait for the next interrupt which will be a C event. Now, the software has memorized that the motor is in its first step in auto-switched mode, so, as we can see in Figure 8, it will enter the Set_ki_autosw routine (Figure 13).

4.7 AUTO-SWITCHED MODE PART OF C INTERRUPT ROUTINE

Figure 13. Commutation event routine (first step in auto-switched mode)

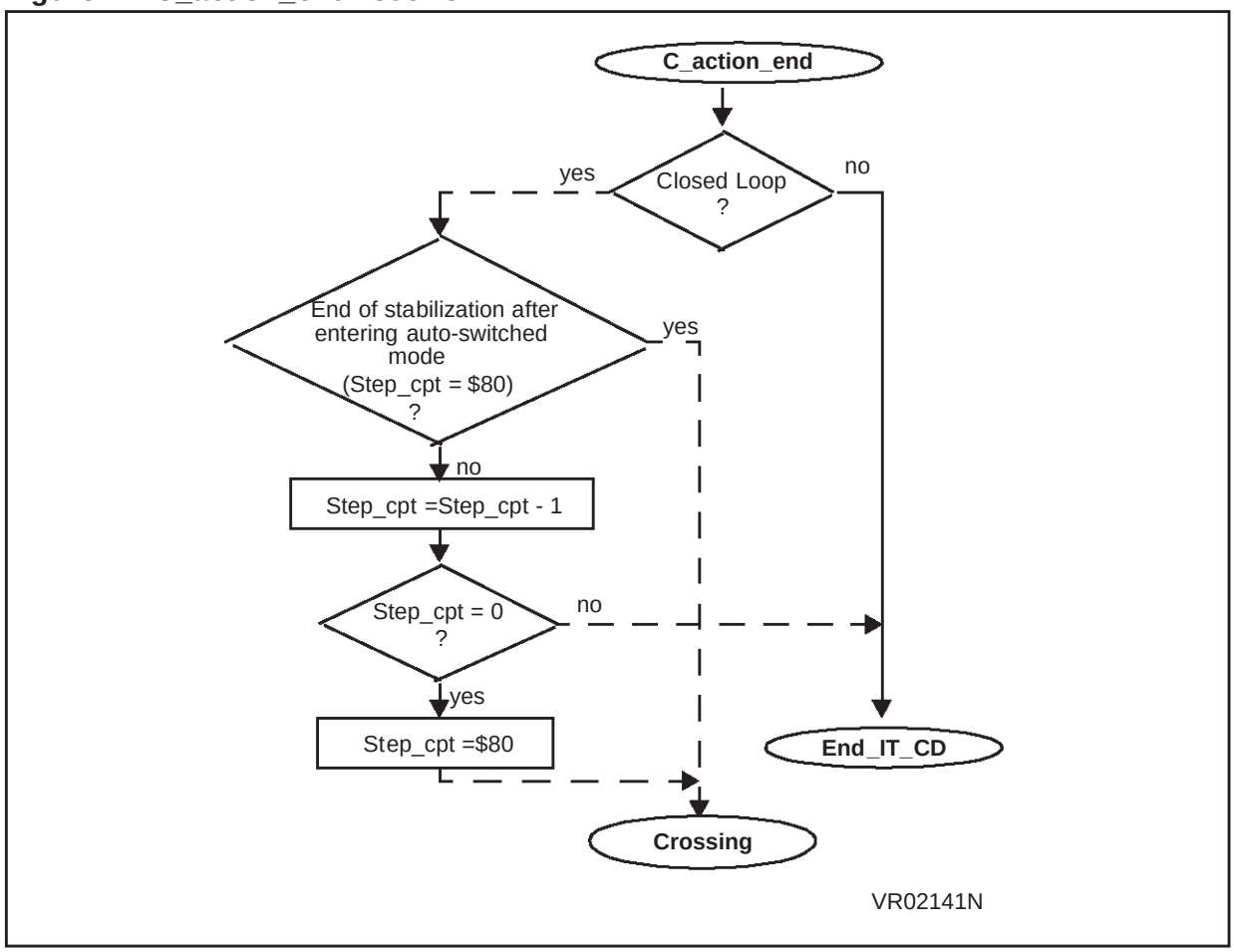


Auto-switched mode is enabled. The MWGHT register is loaded with the delay coefficient meaning that now, the MCOMP register will contain the value of the real delay (time in each step between the zero crossing event and the next commutation event, see Application note 1082).

The Z event time is memorized and then depending on the step configuration (falling or rising edge of the end of Demagnetization event, see Application note 1130), there will be a software or a hardware Demagnetization. Software Demagnetization is a special case that is described in more detail in Section 5.2 of this document (Figure 18 and Figure 19). If there is a hardware Demagnetization, the software enters the End_C_action routine where the data for the next commutation event are prepared. The software then goes to the C_action_end routine (Figure 14).

4.8 LAST PART OF C INTERRUPT ROUTINE

Figure 14. C_action_end routine



In this routine, execution depends on whether we are running in open loop or closed loop. If the motor is running in closed loop (speed regulation), the software checks if the stabilization phase is finished (fixed number of steps in auto-switched mode before entering the speed regulation loop).

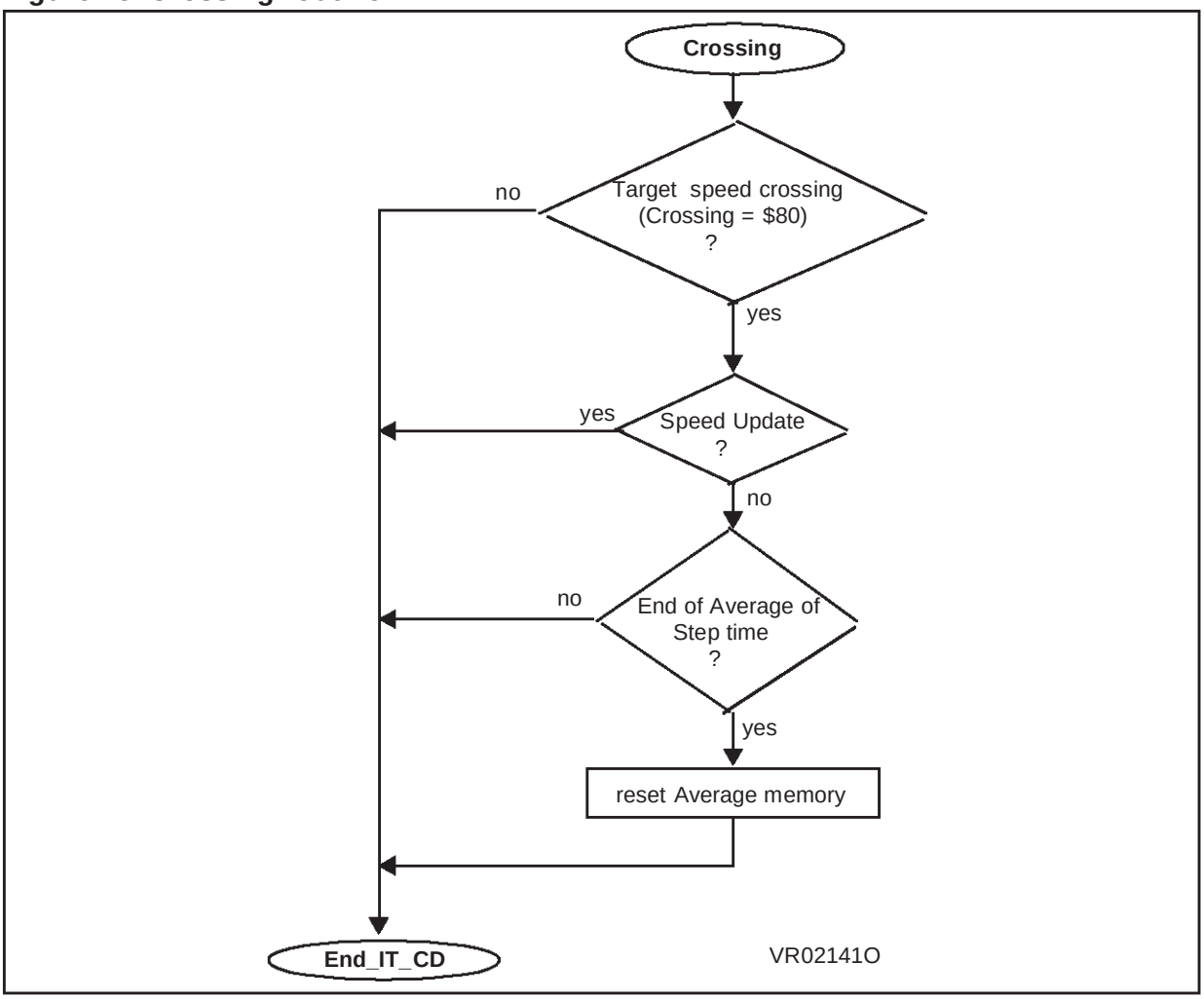
If the stabilization phase is not finished, the count of the number of steps in auto-switched mode is incremented and the software goes back to the End_IT_CD routine (Figure 7) and then goes back to the main program.

If the stabilization phase is finished, the software goes to the crossing routine (Figure 15) to check if the target speed of the motor has been crossed.

If the motor is running in open loop, the software just exits the C interrupt routine.

Note that the number of steps for the stabilization phase is fixed by the software programmer in a RAM variable. Step_cpt acts like a flag and $80 is an arbitrary value decided by the software programmer.

Figure 15. Crossing routine



The software checks in this routine if the motor has already crossed the target speed. If not, the software exits the C interrupt routine. This routine is available only in closed loop.

If yes, then the software checks if the speed has to be updated or if the step time average has to be done by the speed regulation subroutine and then goes back to the main program.

As the motor is now in auto-switched mode, the path followed by the software will be always the same:

– Main program waiting for an interrupt

– C event interrupt, the software enters the C_action_autosw (Figure 13) routine then goes back to the main program

– D event interrupt, nothing is done in the software

– Z event interrupt, nothing is done in the software on a Z event in auto-switched mode

However, we have seen that C, D and Z events are not the only events that provoke an interrupt. In auto-switched mode, R (ratio) event, E (emergency stop) event and O (delay multiplication overflow) event generate interrupts. These cases and some others like software Demagnetization and setting the carry bit after a computation are managed in the software. They are described in the last part of this document.

5 OTHER FEATURES

5.1 R INTERRUPT (MTIM TIMER RATIO CHANGE)

There are two types of R event interrupt (see application note 1082), R+ event and R- event. When a Z event occurs while the MTIM timer counter is between 55h and FFh the event is processed normally.

If the timer reaches FFh before the Z or D event occurs, then there is an R+ event (timer overflow). The clock needs to be slowed down, so the ratio is increased.

If the Z event occurs before the timer has reached the value 55h, then there is an R- event (timer underflow). The clock needs to be accelerated, so the ratio is increased.

Figure 16 and Figure 17 show what is done by the software when R+ or an R- event occurs.

If we go back to Figure 10, we can see that, on an R event, the software checks the interrupt flags to determine if it is an R+ or an R- event.

If RPI flag is set, it is a R+ event and the software goes to the Timer_ovf routine (Figure 17).

If RMI flag is set, it is an R- event and the software goes to the Z_55 routine (Figure 16).

Figure 16. Z_55: internal 8-bit timer underflow

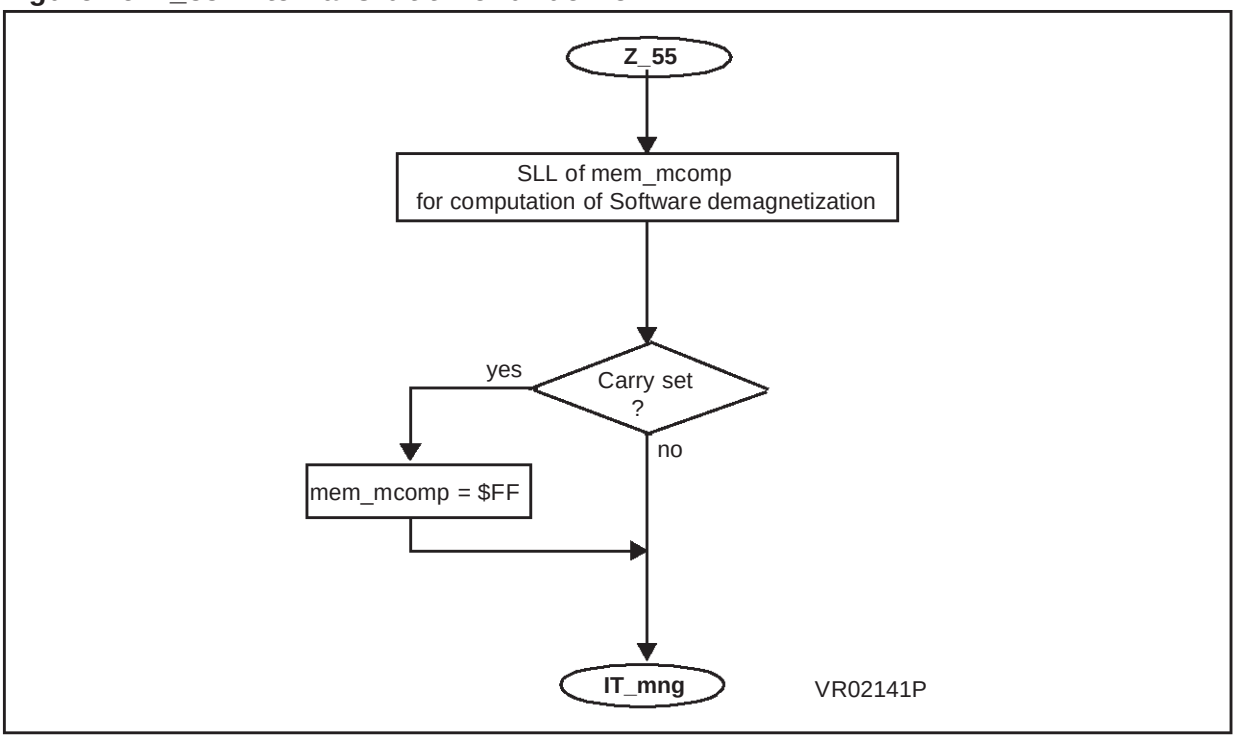


Figure 16 shows what is done in the case of a timer underflow.

In practice, the timer ratio is decreased automatically by the microcontroller and so all the values are multiplied by two for the current step in order to avoid computation errors.

This means that the Zn and Z previous times are multiplied by 2, as is the Demagnetization time value in the MDREG register.

If the carry is set during the computation, the maximum value is stored in the MCOMP register.

Then the software goes back to the IT_mng routine (Figure 10), checks again for the interrupt flags and goes back to the main program and waits for another interrupt.

Figure 17. Timer_ovf: internal 8-bit timer overflow

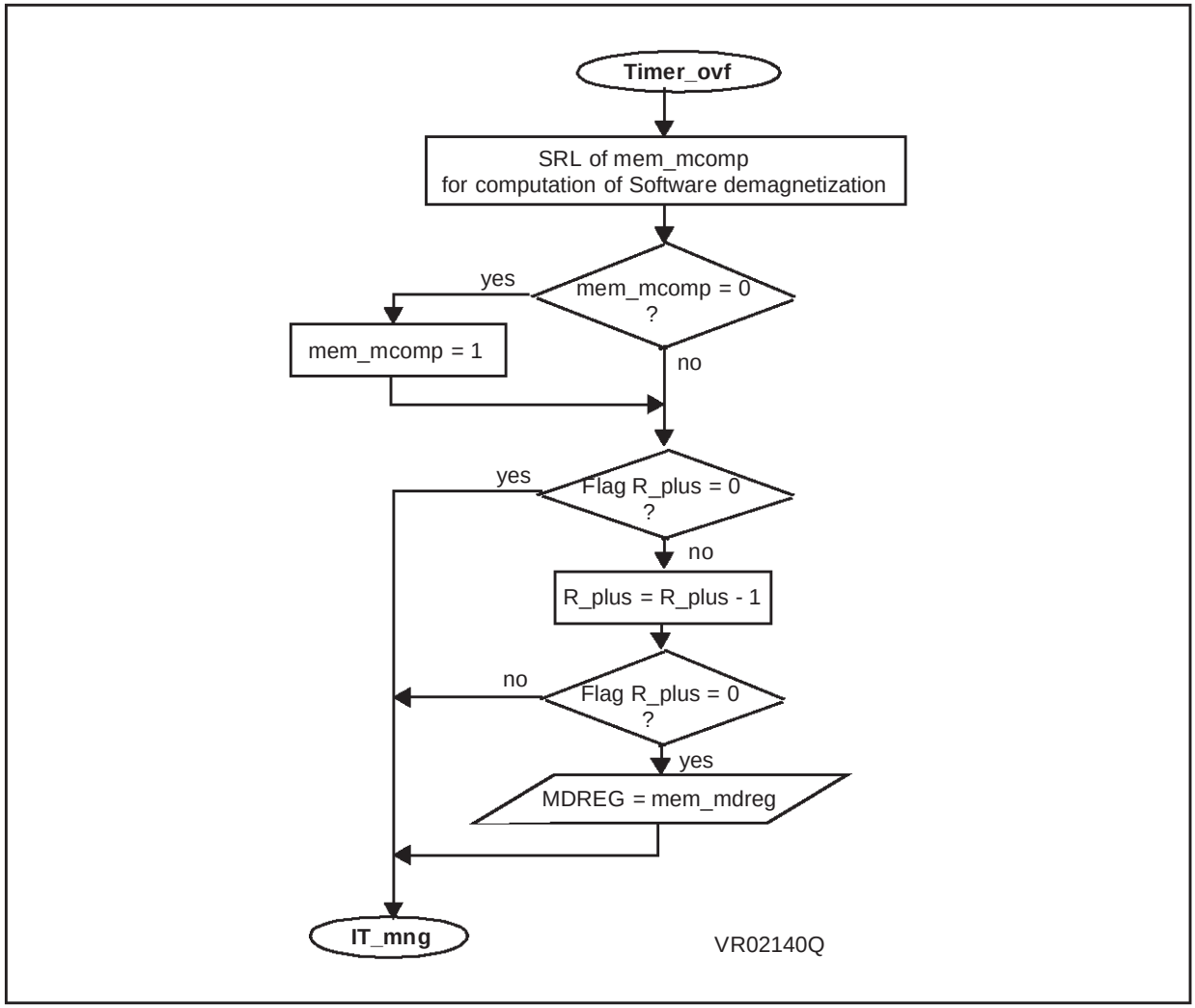


Figure 17 shows what is done when a timer overflow occurs.

In practice, the timer ratio is increased automatically by the microcontroller and so all the values are divided by two for the current step in order to avoid computation errors.

This means that the Zn and Z previous times are divided by 2, as is the Demagnetization time value in the MDREG register.

The timer is reset to its middle value 7Fh

If, after division, the value in the MCOMP register is equal to zero, the software stores a minimum value of 1 in it.

Then the software checks again for the R+ event flag and goes back to the IT_mng routine (Figure 10).

5.2 SOFTWARE DEMAGNETIZATION

When auto-switched mode is enabled, in this software, every second step uses software demagnetization. Note that this is a software example and that the fact that every second step uses software software demagnetization can be changed.

So, at each step, the software checks if it is a step with a hardware or a software demagnetization (set by the HDM or SDM bits in the MCRB register). We can see this on Figure 13.

Figure 18 and Figure 19 show what is done in the software in the case of a software demagnetization.

Figure 18. Software demagnetization (first part)

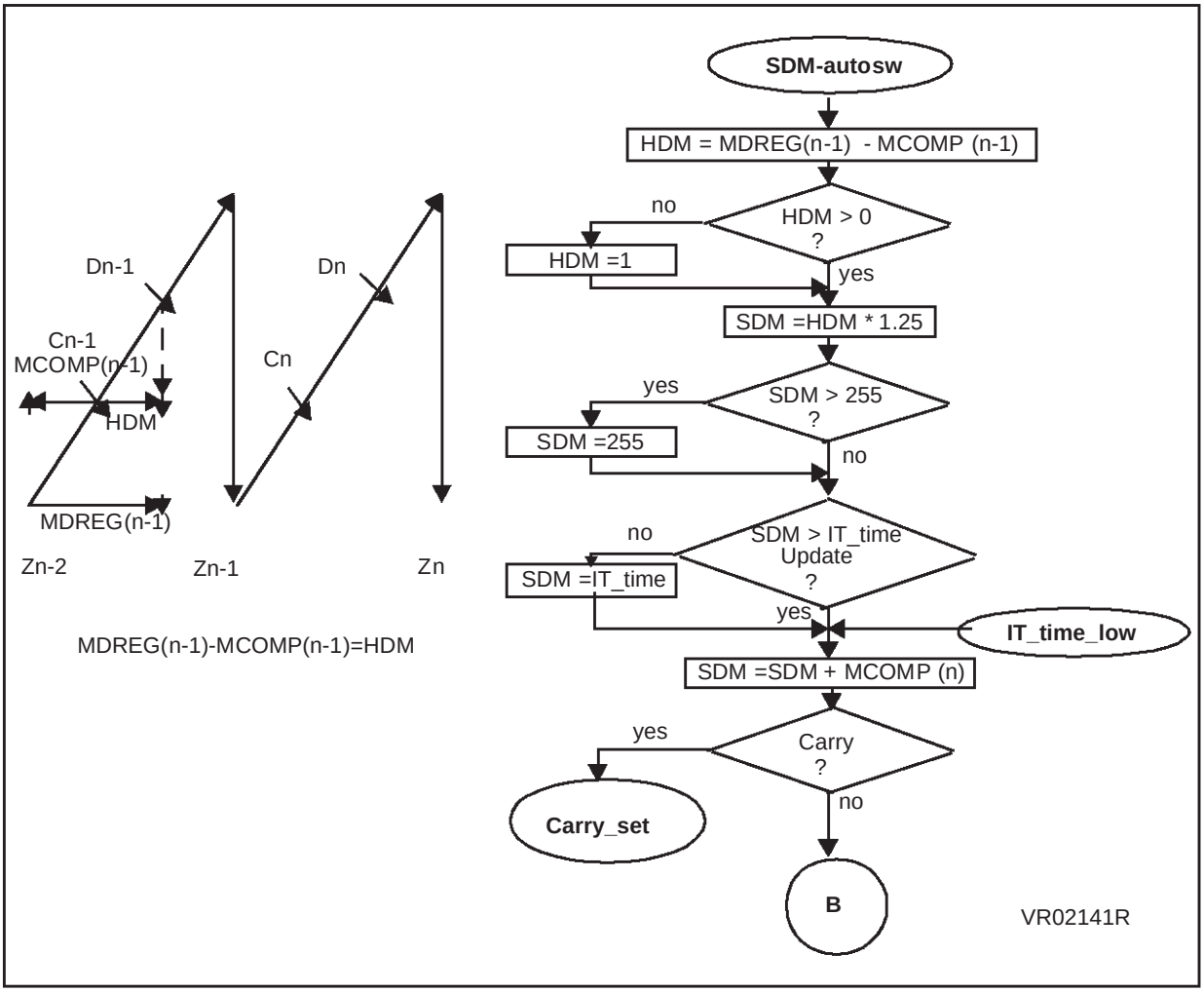


Figure 18 shows the first part of the software demagnetization.

Let's assume that we will have a software demagnetization in the current step (step n). This means that the step before, it was a hardware demagnetization (step n-1). Dn-1 was a hardware demagnetization.

The values in the MDREG (n-1) registers for the hardware demagnetization and MCOMP(n-1) for the computed delay have been memorized.

If we look at the values in register MDREG (n-1), this value represents the time between the previous Z event (Z n-2) and the end of demagnetization event at step n-1 (Dn-1).

The real hardware demagnetization time is the time between the commutation event (n-1) and the end of demagnetization event (n-1). So, the first thing done by the software is to compute the real hardware demagnetization event from the previous step, this value is HDM:

HDM (n-1)=MDREG(n-1)-MCOMP(n-1)

Then the software checks if this value is greater than 0. If not, a minimum value of 1 is set. Then, to compute the software demagnetization time for the current step (step n), the hardware Demagnetization time is multiplied by a correction factor (chosen arbitrarily by the software programmer) to be sure that the software Demagnetization time for the current step will be greater than the hardware demagnetization time from the previous step (for safety purposes to avoid the end of Demagnetization event occurring too early).

SDM (software Demagnetization value)=HDM*1.25

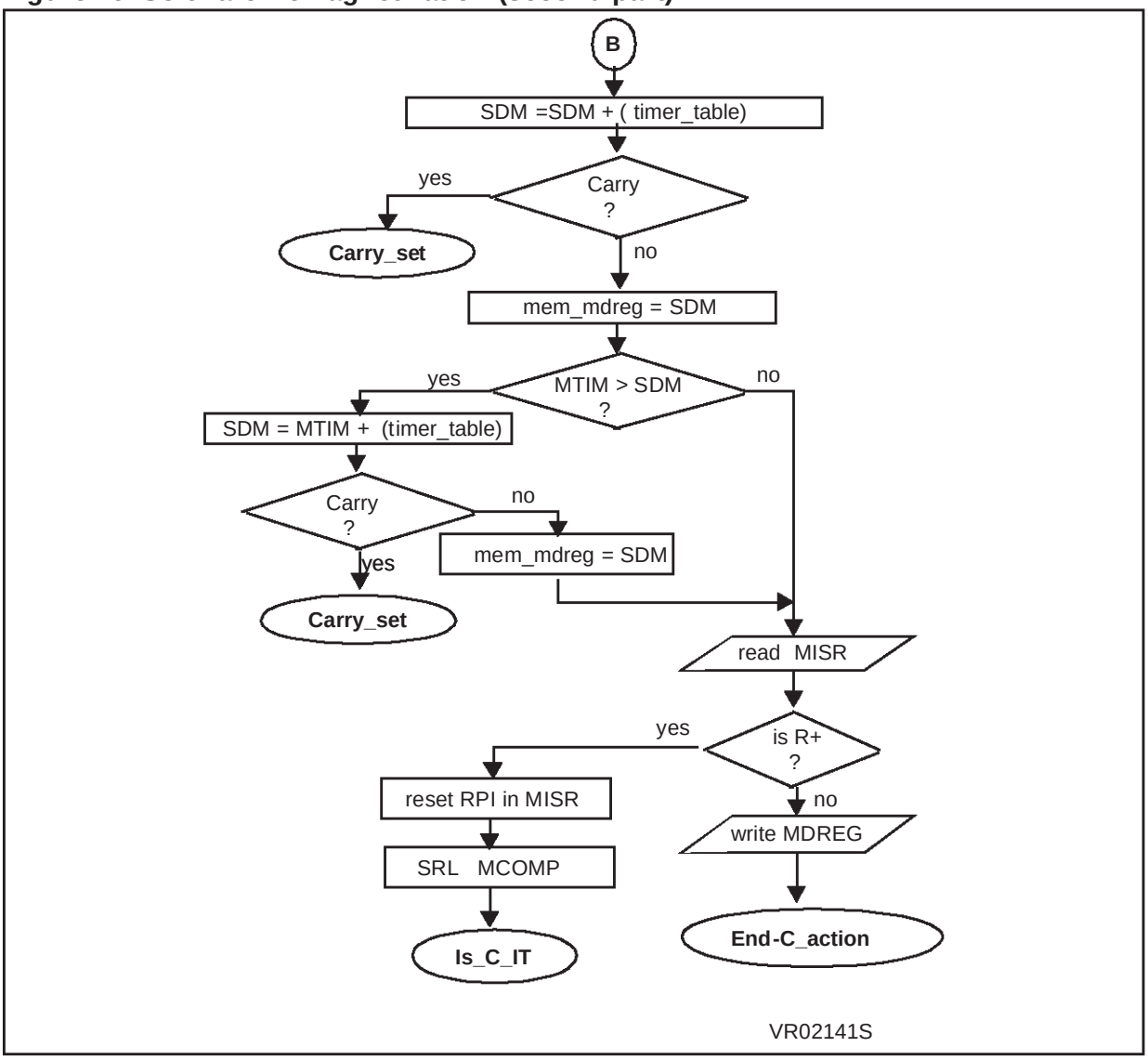
Then the software checks if this value is greater than 255. If yes, this value is forced to 255.

Then the software checks if the value of the software Demagnetization time is greater than the time needed to process the interrupt routine (IT_time). If not, we put this interrupt processing time as a minimum value. This is done because, if once the routine is finished the value in the internal 8-bit timer is greater than the value of the software Demagnetization, the end of Demagnetization will never happen, as the value of the software Demagnetization would have been already reached by the internal timer before the end of the routine.

Then, at this value is added the value of the delay between the Zn-1 event and the next commutation event Cn, in order to get the time between the Zn-1 event and the software End of Demagnetization event.

If the carry has been set by these computations, the software goes to the carry-set routine (Figure 20), if not, the software Demagnetization routine continues (Figure 19).

Figure 19. Software Demagnetization (second part)



The software adds another correction factor to the software Demagnetization time corresponding to the time needed to load the register. If the carry has been set, we go to the carry_set routine. If not the actual value for the software end of Demagnetization event is memorized and the software checks if this value is greater than the current value of the MTIM timer.

If yes, a greater value than in the MTIM timer is put for the software Demagnetization time.

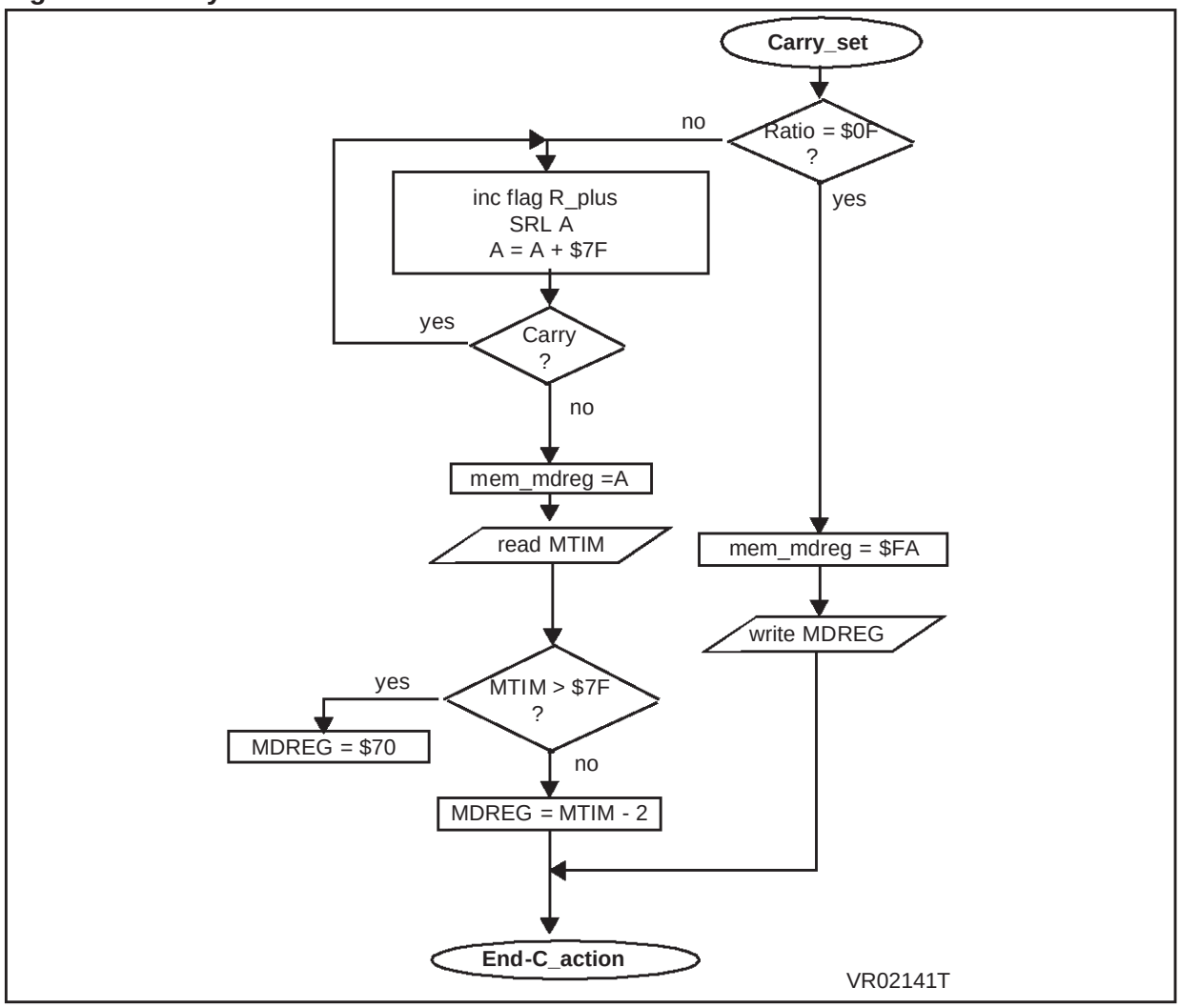
If not, the software checks if the flag for R+ event is set

– If yes, the flag is reset and the software goes back to the beginning of the C interrupt routine

– If not, the value is written in MDREG register, then the software goes to the End_C_action routine to prepare the data for the next C event and goes back to the main program

If the carry has been set during the computation for the software end of Demagnetization event, then the software enters the carry_set routine shown in Figure 20.

Figure 20. Carry set



If the carry has been set, it means that the software end of Demagnetization event will happen after the time the MTIM timer would have reached the value FFh. There will be an overflow of the MTIM timer (R+ event). So, in the software, we check if the ratio of the MTIM timer is at the maximum.

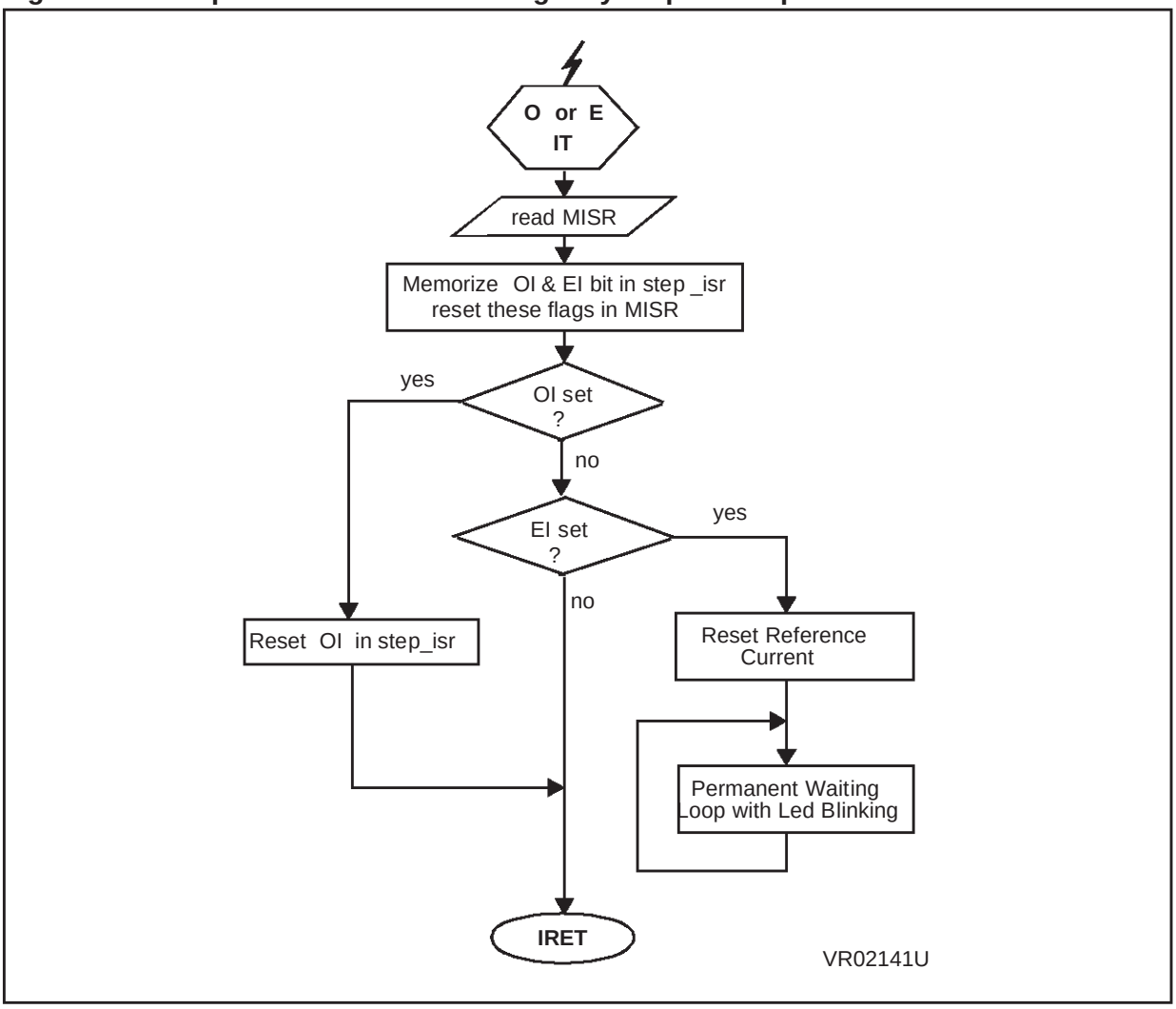
– If not, the R_plus flag is incremented, meaning that the overflow will happen for the software Demagnetization. The clock needs to be slowed down, so the ratio needs to be incremented and to update the value to the new future ratio, the value for the software Demagnetization is divided by 2. Then, when the ratio changes, the MTIM timer is reset to its middle value, it is 7Fh, so, we add 7Fh to the software Demagnetization value. This value is then memorized in the variable mem_mdreg and the MTIM timer is read. To be sure that no end of Demag-

netization will happen before the ratio change (when the R+ event will happen), a lower value than the MTIM value is put in the MDREG register. The next interrupt will be a R+ event. At that time, the MDREG register is written.

– If yes, an arbitrary value is memorized for the software end of Demagnetization event and the register MDREG is written.

5.3 O (MULTIPLIER OVERFLOW) AND E (EMERGENCY STOP) INTERRUPT ROUTINES

Figure 21. Multiplier overflow and emergency stop interrupt routine



If there is a multiplication overflow when computing delay between the zero crossing event and the next commutation, nothing is really done by the software, the interrupt flag is reset and the software goes back to the main program.

For the emergency stop interrupt, the reference current is reset and the software enters a permanent loop, waiting for the end of the emergency stop signal.

6 SUMMARY

Figure 22. General overview of the software

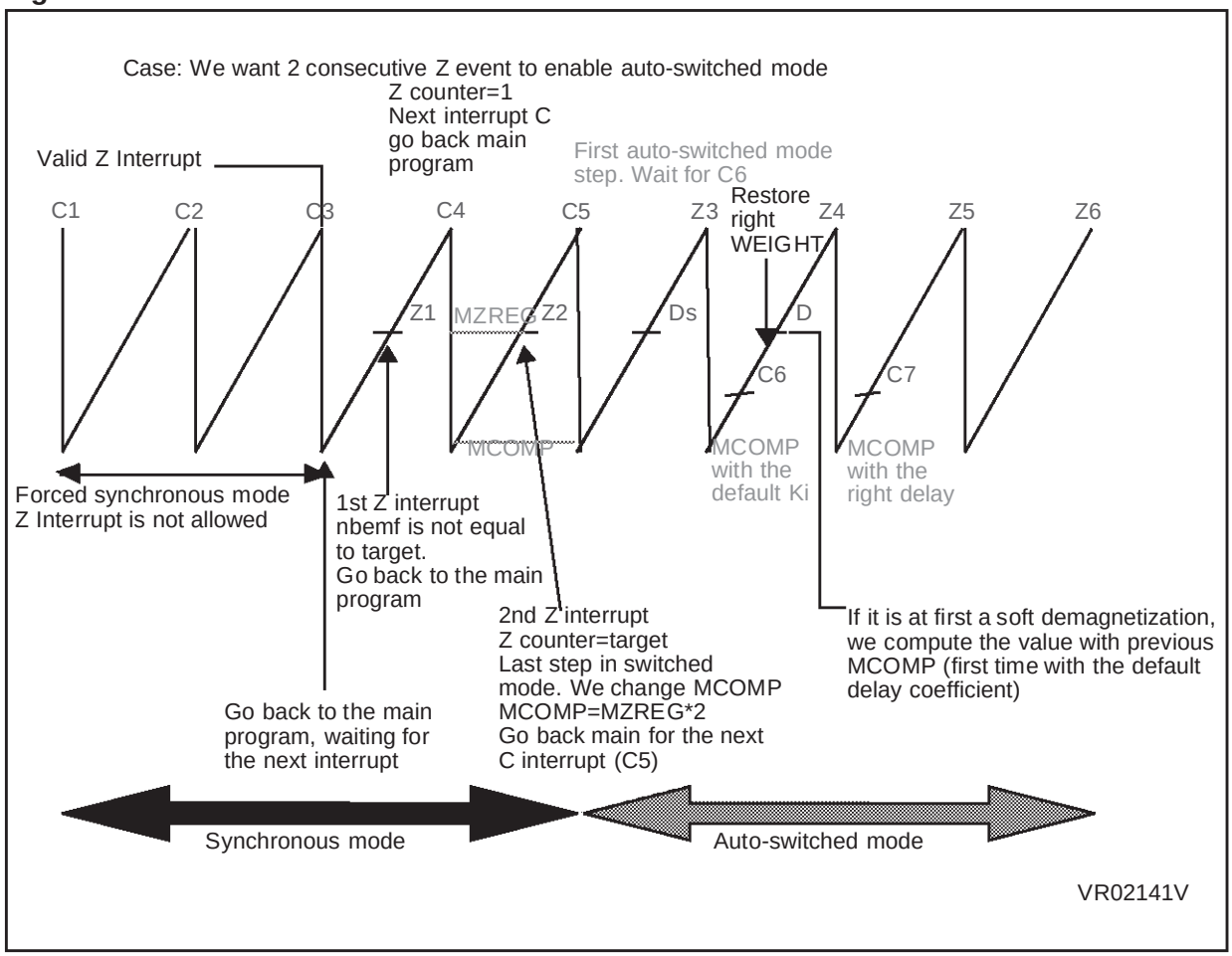


Figure 22 shows how the software behaves in the starting sequence, when the motor is passing from synchronous mode (switched mode) to auto-switched mode. This figure illustrates the case where the software programmer wants 2 consecutive Z events before enabling auto-switched mode and 3 steps in forced synchronous mode.

At C3, the Z event interrupt is enabled, then the software is waits for the first Z event and then for 2 consecutive Z events before enabling auto-switched mode.

When Z2 occurs, the target number of consecutive Z events is reached. Z2 is centered on the current step and the software memorizes that it is the last step in synchronous mode.

The step after, the software memorizes that it is the first step in auto-switched mode and the MTIM timer is now reset on the Z event. The D event interrupt is enabled. Note that during all the steps in synchronous mode, the D event interrupt is not enabled and all the End of Demagnetization events are software D events.