AP16109

XC164CM Easy Kit + BLDC Drive Application Kit

XC164CM+BLDC-Motor "Cookery-Book" without DAvE-DRIVE - just to gain Know-How about "How to create a BLDC-Motor application (Hall Sensor Mode)". You can do all programming examples in this document with the evaluation version of the KEIL compiler.

Microcontrollers

Never stop thinking

Edition 2008-07-16

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP16109		
Revision History:	2006-12	V2.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of contents:

Introduction:

This “Appnote” is an Infineon Hands-On-Training.
It will help inexperienced users to get the BLDC-Drive-Application-Kit-BTS-7960 together with the XC164CM-Easy-Kit up and running.

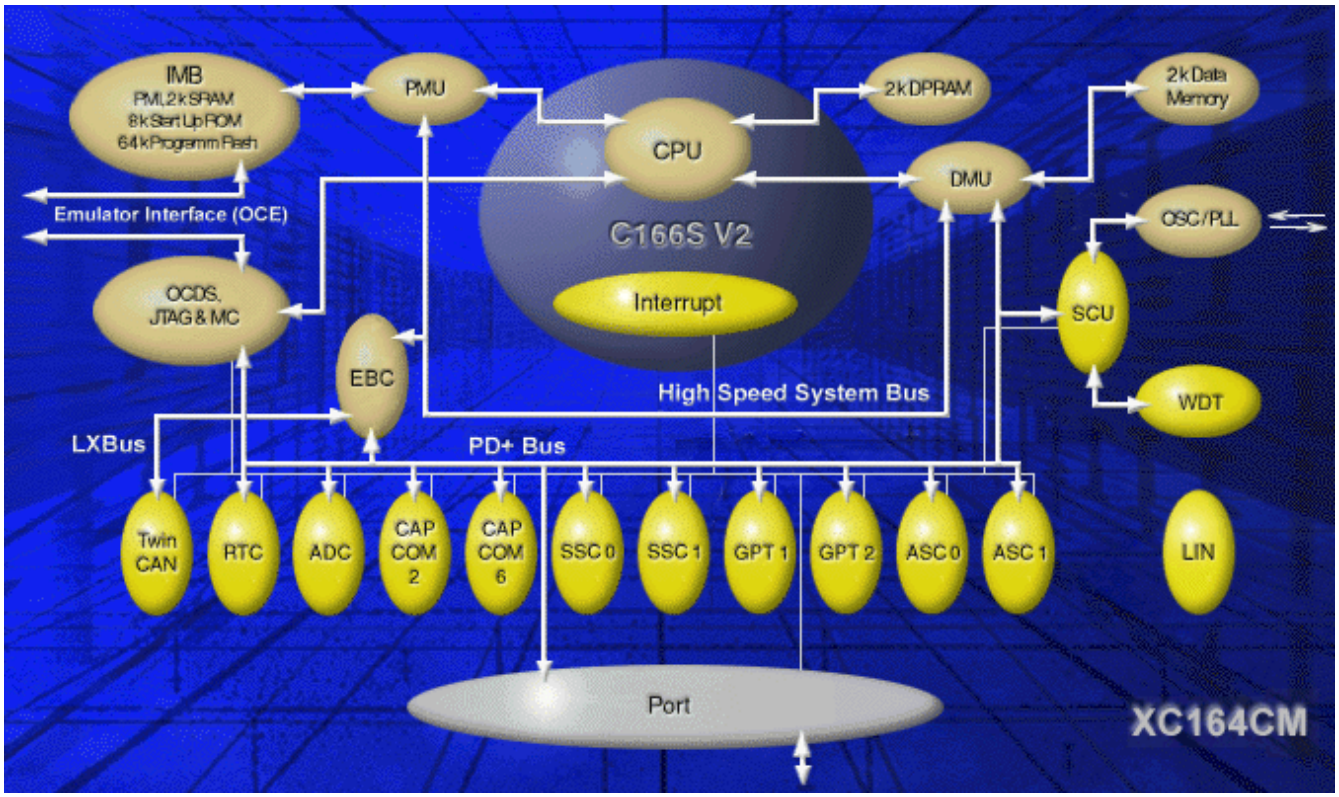
With this Hands-On-Training / Cookery-Book / step-by-step-book you should be able to get your BLDC-Motor running in less than 4 hours.

The purpose of this Appnote is to provide you with information about the requirements for creating a BLDC-Motor application using the CAPCOM6-Module.

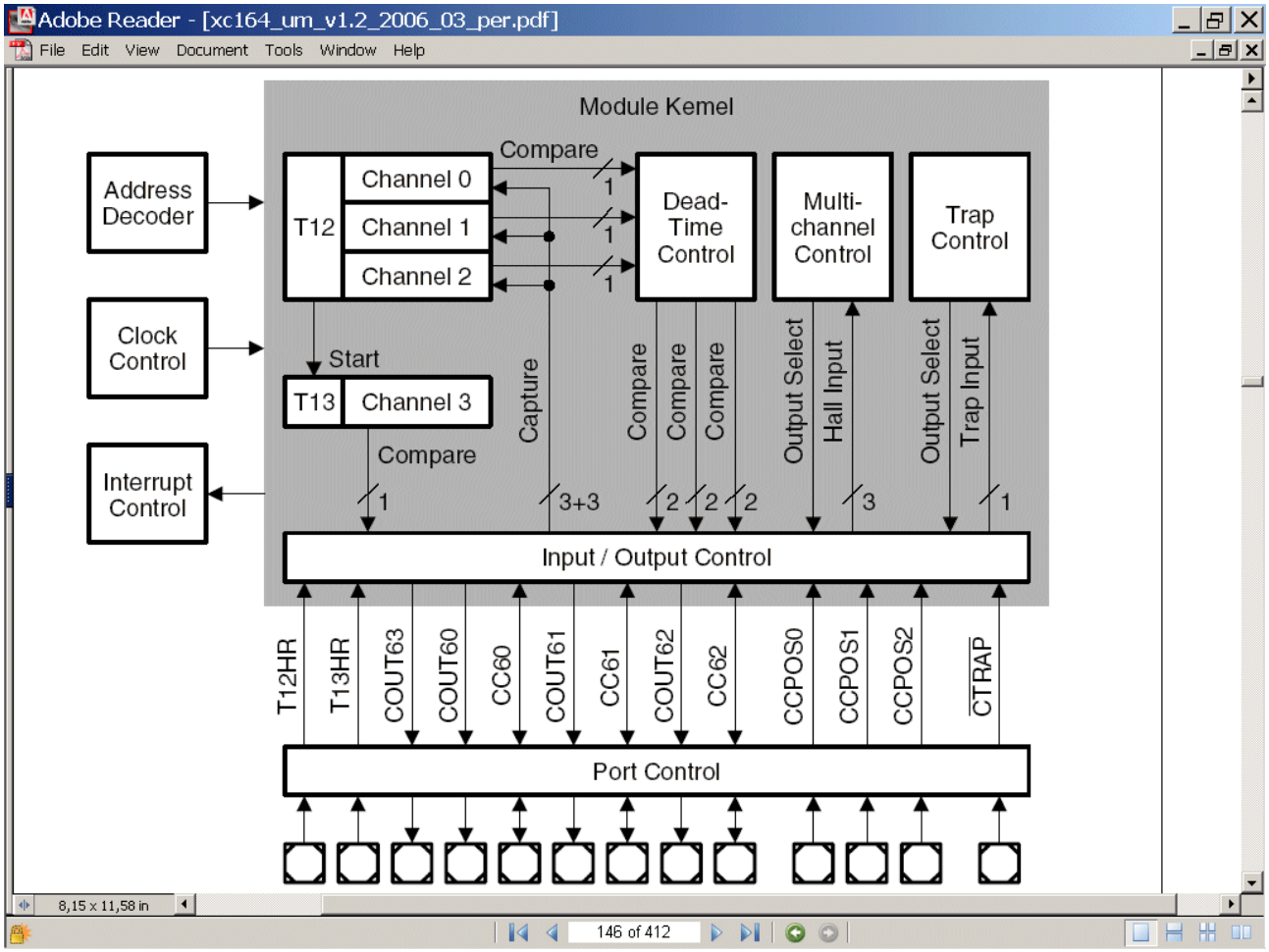
Note:
The style used in this document focuses on working through this material as fast and easily as possible. That means there are full screenshots instead of dialog-window-screenshots; extensive use of colours and page breaks; and listed source-code is not formatted to ease copy & paste.

Have fun and enjoy the running motor!

XC164CM Block Diagram (used Microcontroller):



CAPCOM6 Block Diagram (used Modul for PWM generation):

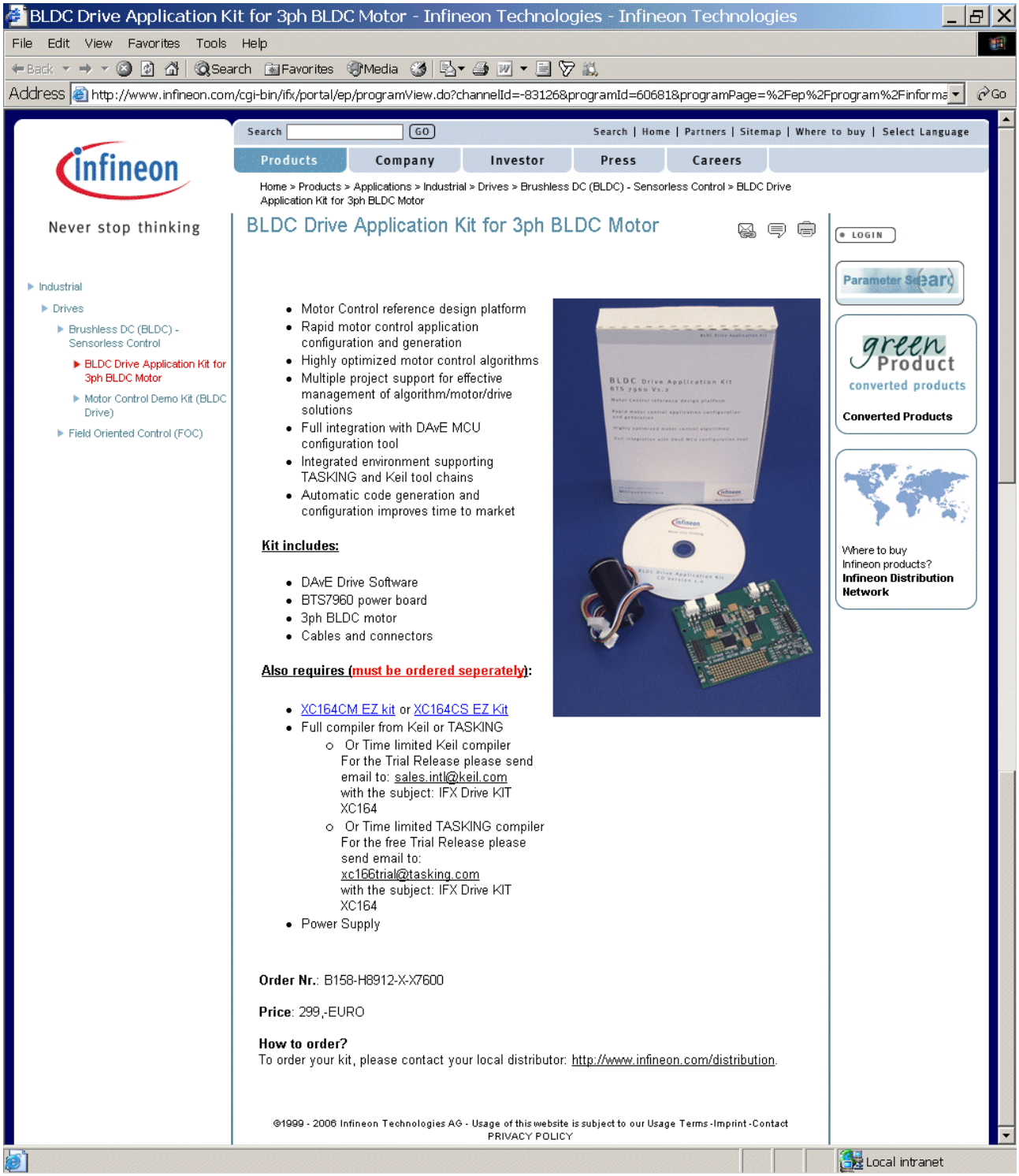


You need the XC164CM Easy Kit:

http://www.infineon.com/cgi-bin/ifx/portal/ep/programView.do?channelId=-65582&programId=42245&programPage=%2Fep%2Fprogram%2Finformation.jsp&pageTypeId=17099&BV

You need the BLDC Drive Application Kit:

http://www.infineon.com/cms/en/product/applications/industrial/microcontrollers/drives/bldc/bldc-app-kit.html

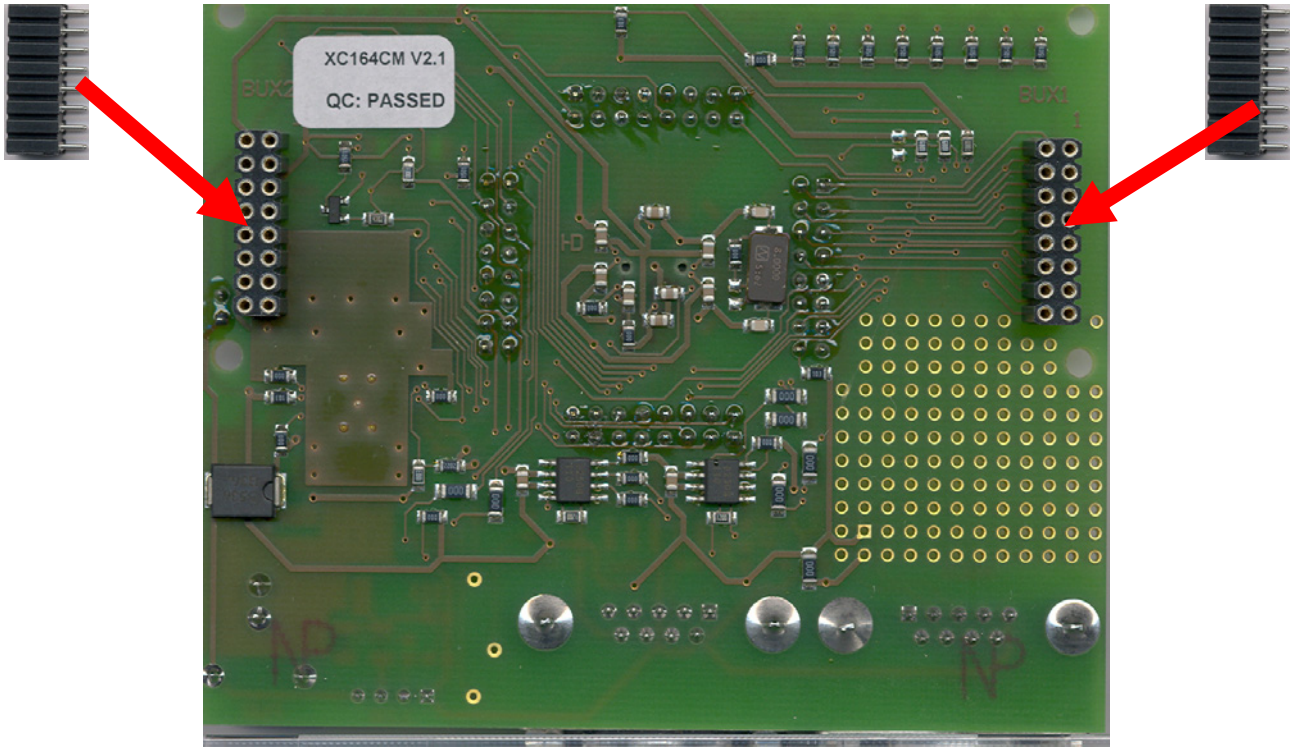


Preparing the Hardware:

You need a soldering iron with a small solder tip.

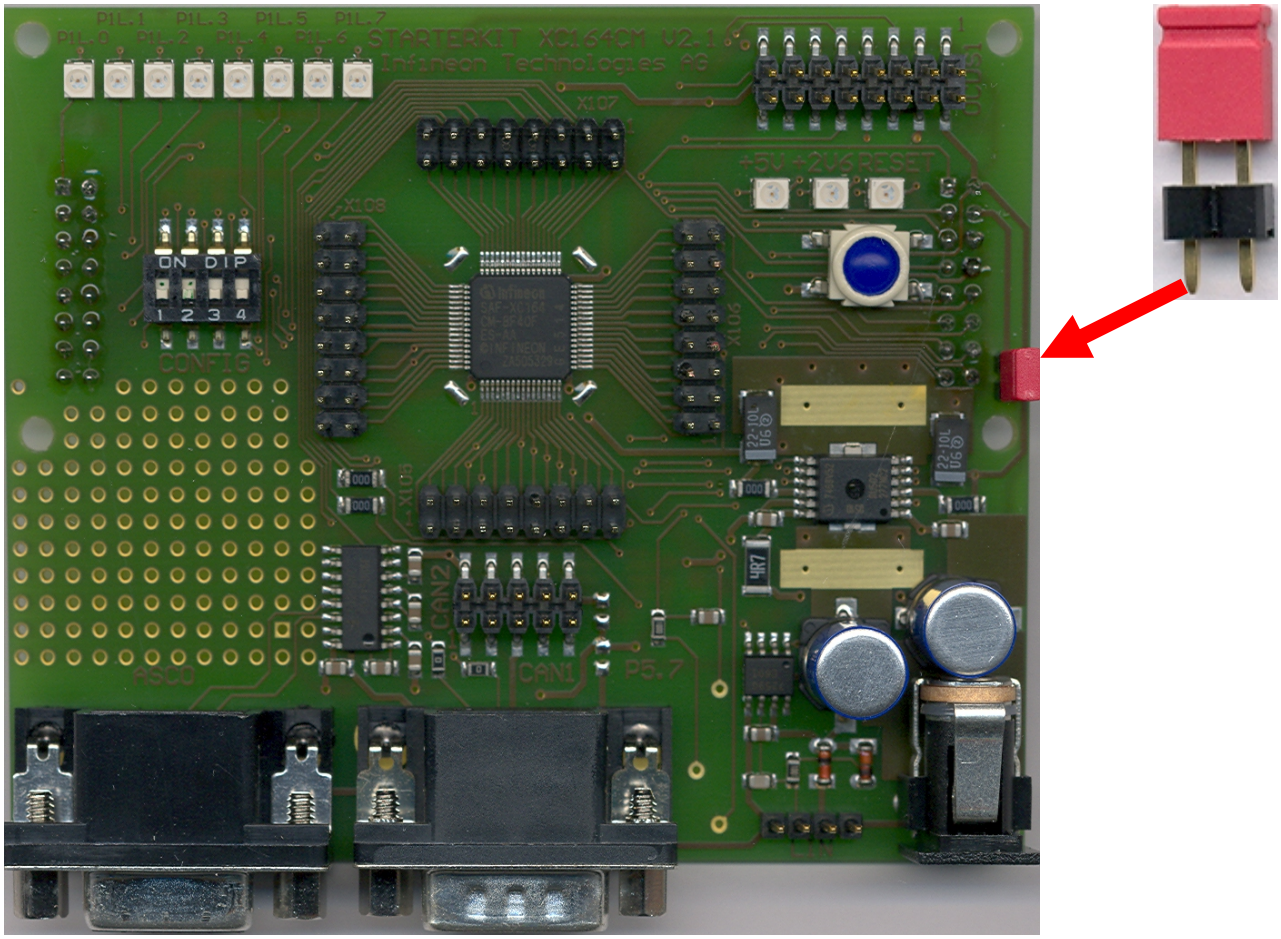
Solder the two Connectors (Accessories in the BLDC Drive Application Kit) onto the back of the XC164CM Easy Kit:

Purpose: Connection between the two boards.

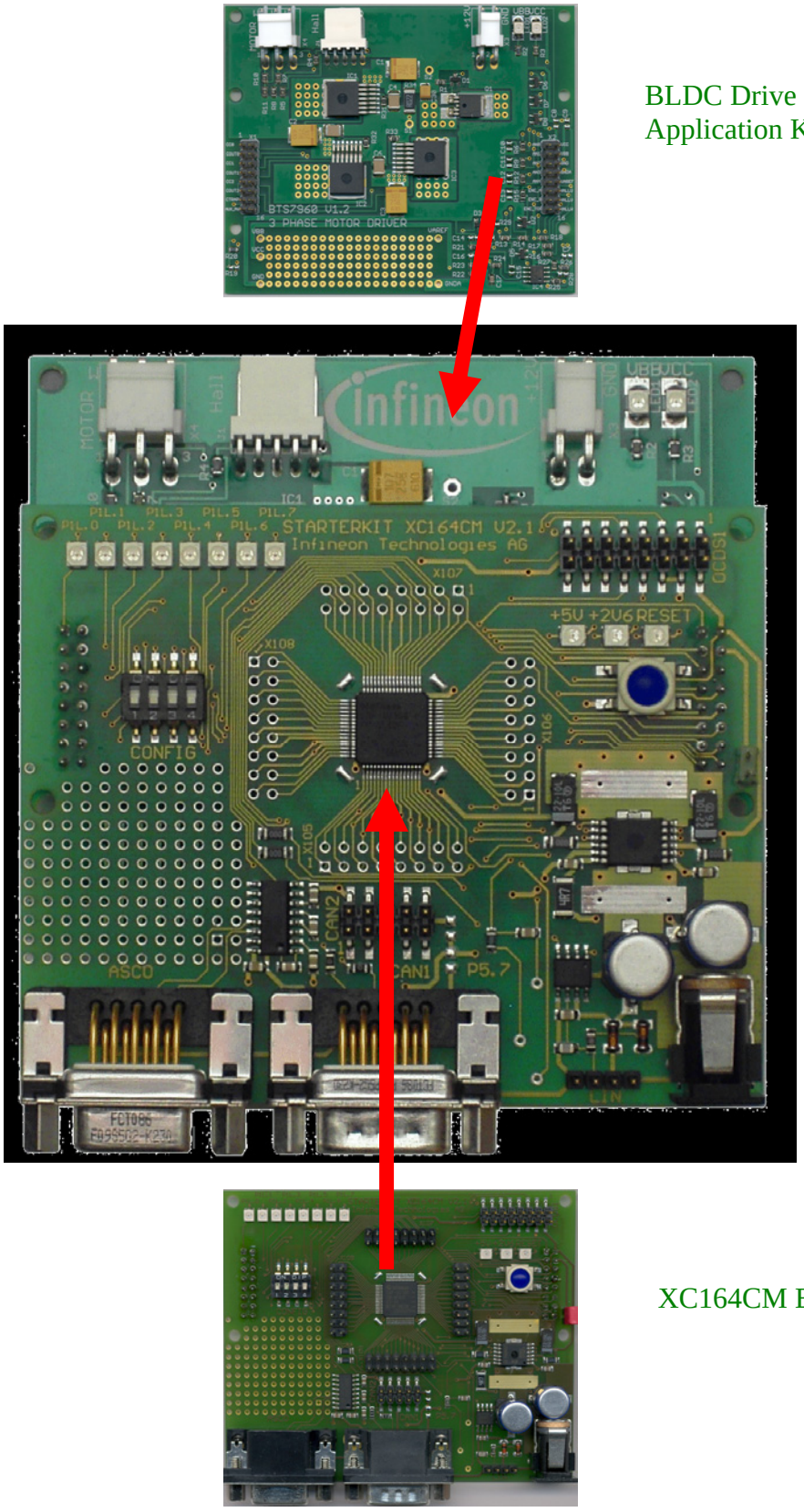


Solder the Jumper (Accessories in the BLDC Drive Application Kit) onto the upper surface of the XC164CM Easy Kit:

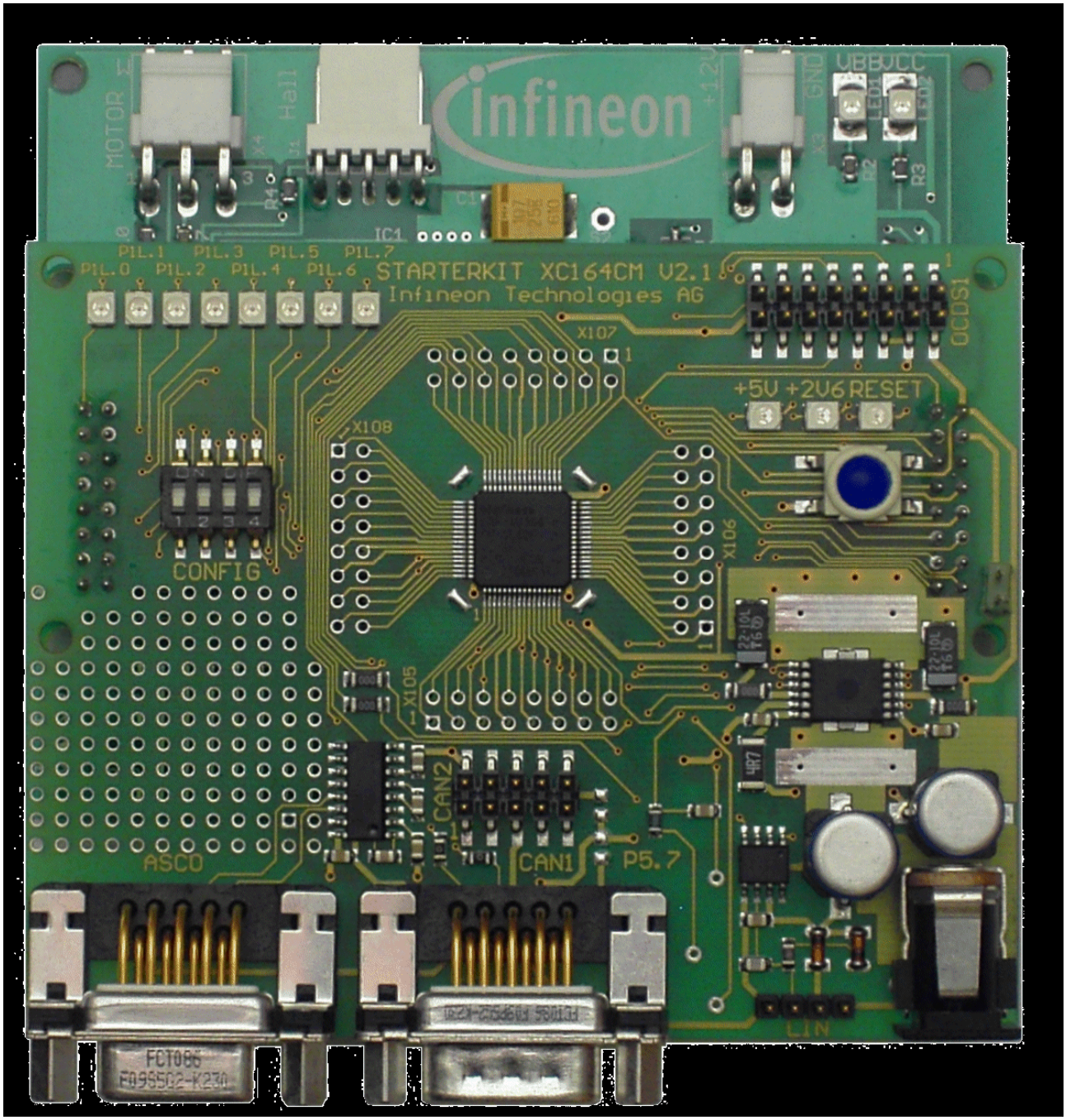
Purpose: Power-Supply from the BLDC Drive Application Kit.



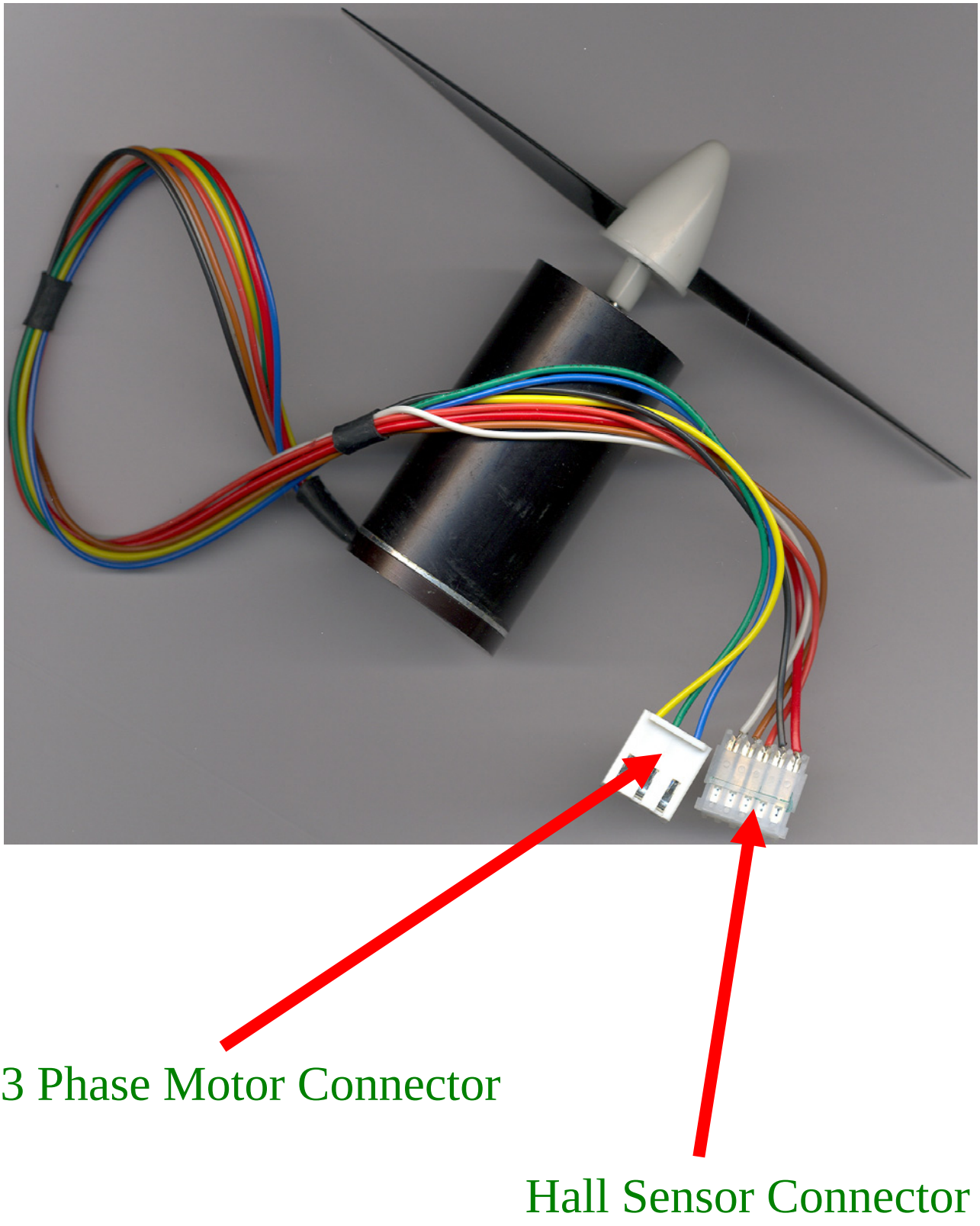
Plug the XC164CM Easy Kit onto the BLDC Drive Application Kit:



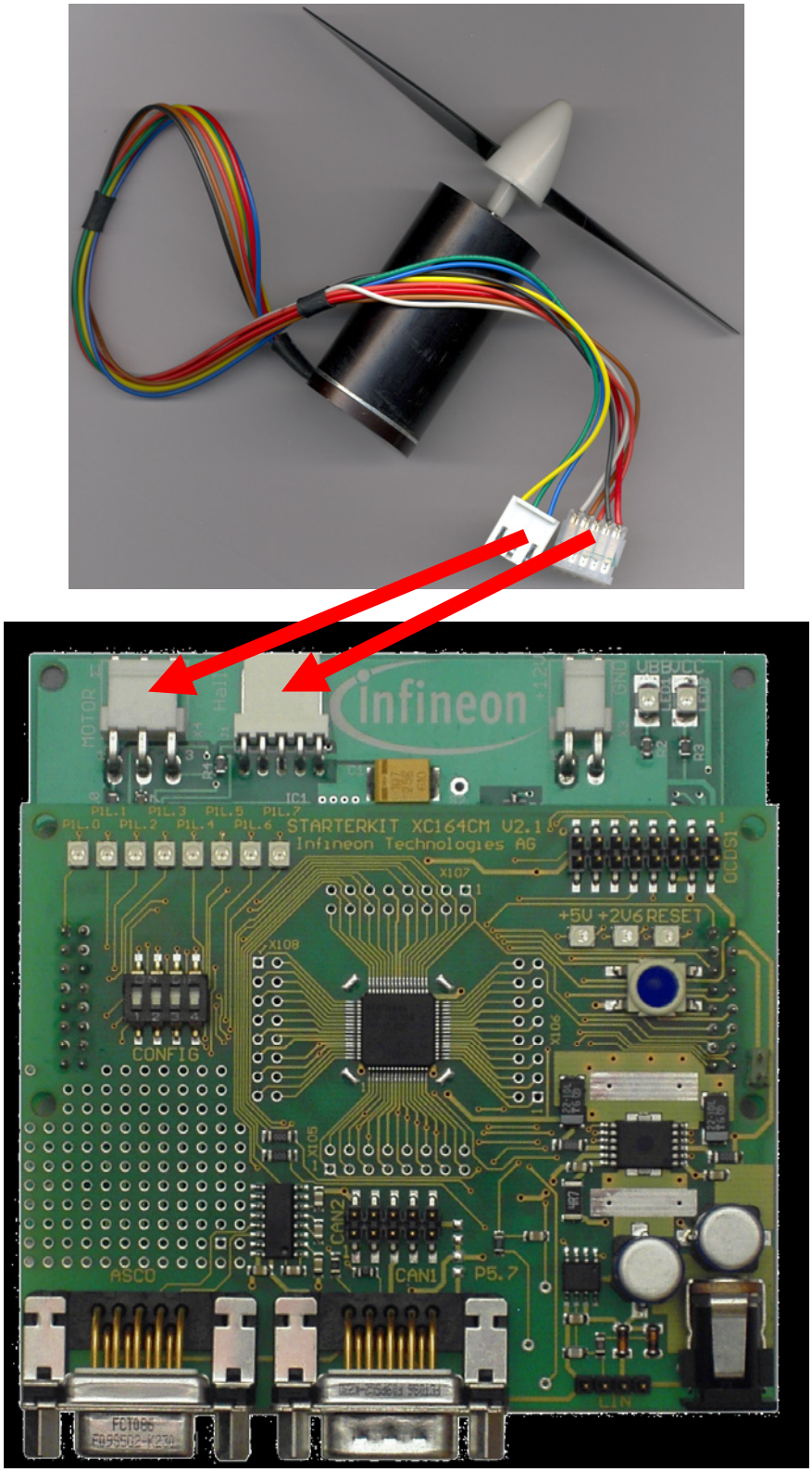
Result:


infineon
MOTOR
Hall
VBBVCC
GND
+1.2V
STARTERKIT XC164CM V2.1
Infineon Technologies AG
P1L.0 P1L.1 P1L.2 P1L.3 P1L.4 P1L.5 P1L.6 P1L.7
X107
X108
X106
X105
OCDS1
+5V +2V6 RESET
CONFIG
CAN2
CAN1
P5.7
ASCO
LIN
FCT086

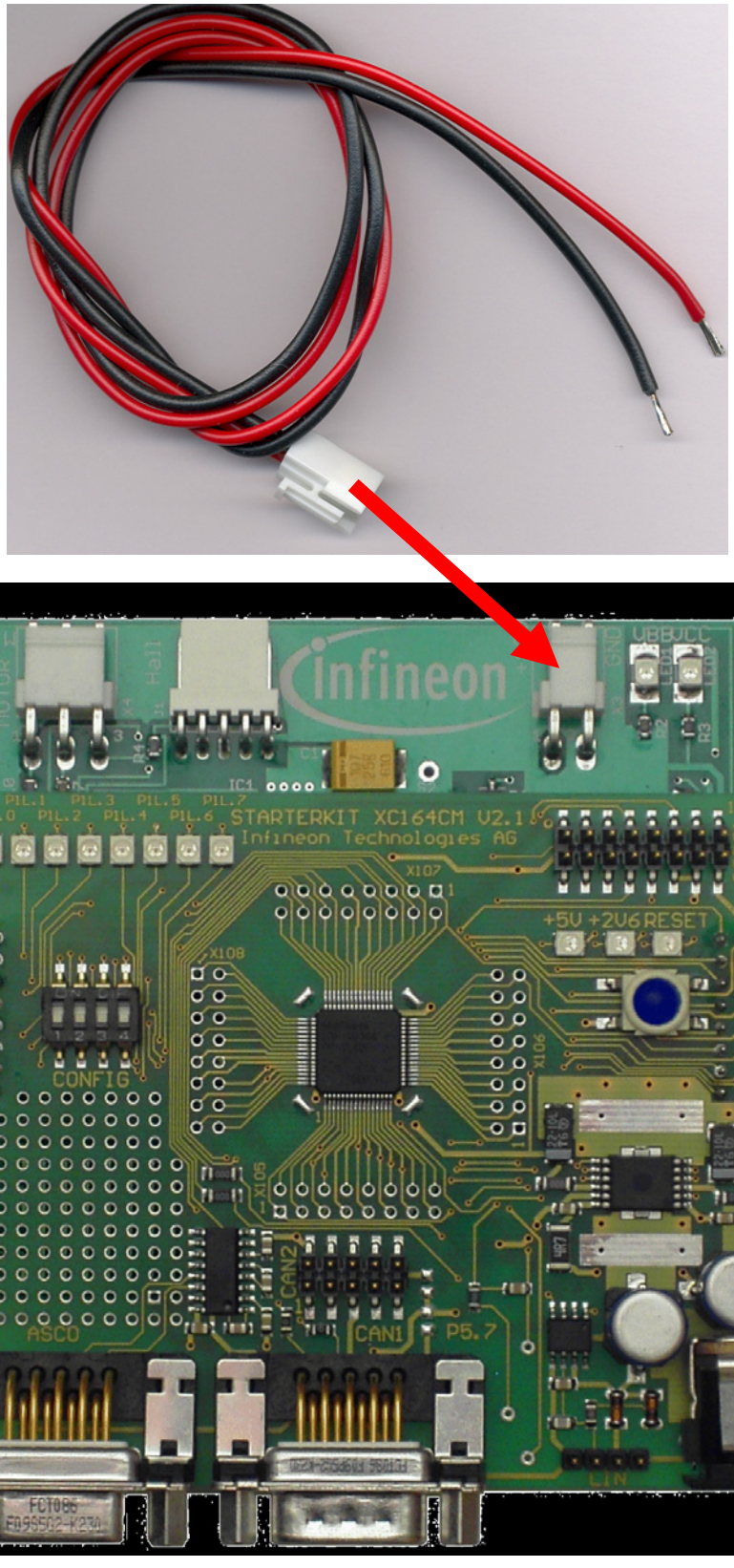
The Motor:

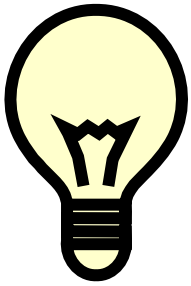


Connect the Motor to the BLDC Drive Application Kit:



Connect the 12 Volt Power Supply to the BLDC Drive Application Kit:





The following pictures (screenshots) of a Powerpoint presentation should give you a first understanding about the requirements for driving a BLDC-Motor in Hall Sensor Mode:

Note:

A brushless DC motor (BLDC) is an AC synchronous electric motor that from a modeling perspective looks very similar to a DC motor. Sometimes the difference is explained as an electronically-controlled commutation system, instead of a mechanical commutation system, although this is misleading, as physically the two motors are completely different.

In a conventional (brushed) DC-motor, the brushes make mechanical contact with a set of electrical contacts on the rotor (called the commutator), forming an electrical circuit between the DC electrical source and the armature coil-windings. As the armature rotates on axis, the stationary brushes come into contact with different sections of the rotating commutator. The commutator and brush-system form a set of electrical switches, each firing in sequence, such that electrical-power always flows through the armature-coil closest to the stationary stator (permanent magnet.)

In a BLDC motor, the brush-system/commutator assembly is replaced by an intelligent electronic controller. The controller performs the same power-distribution found in a brushed DC-motor, only without using a commutator/brush system. The controller contains a bank of MOSFET devices to drive high-current DC power, and a microcontroller to precisely orchestrate the rapid-changing current-timings. Because the controller must follow the rotor, the controller needs some means of determining the rotor's orientation/position (relative to the stator coils.) Some designs use Hall effect sensors to directly measure the rotor's position. Others measure the back EMF in the undriven coils to infer the rotor position, eliminating the need for separate Hall effect sensors, and therefore are often called "sensorless" controllers. (The BLDC motor has a trapezoidal backEMF, while a brushless AC motor has a sinousoidal backEMF.)

In the 'conventional' (also known as 'inrunner') configuration, the permanent magnets are mounted on the spinning armature (rotor.) The stator coils surround the rotor.

BLDC-Motor

Brushless DC-Motor

EC-Motor (Electronic Commutation Motor)

Electronically Commutated DC Motors (Brushless)

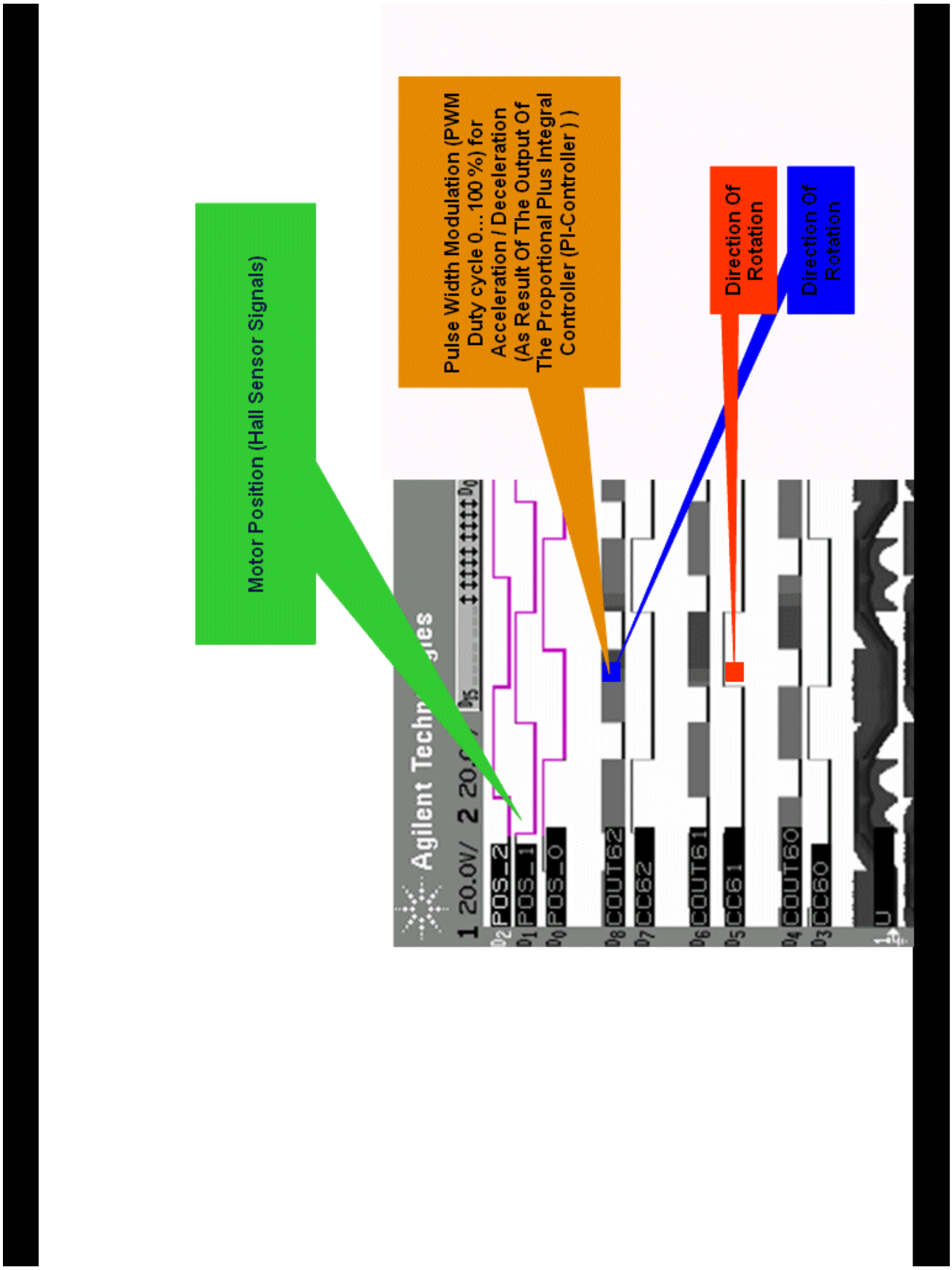
Brushless Synchronous 3-Phase Motor

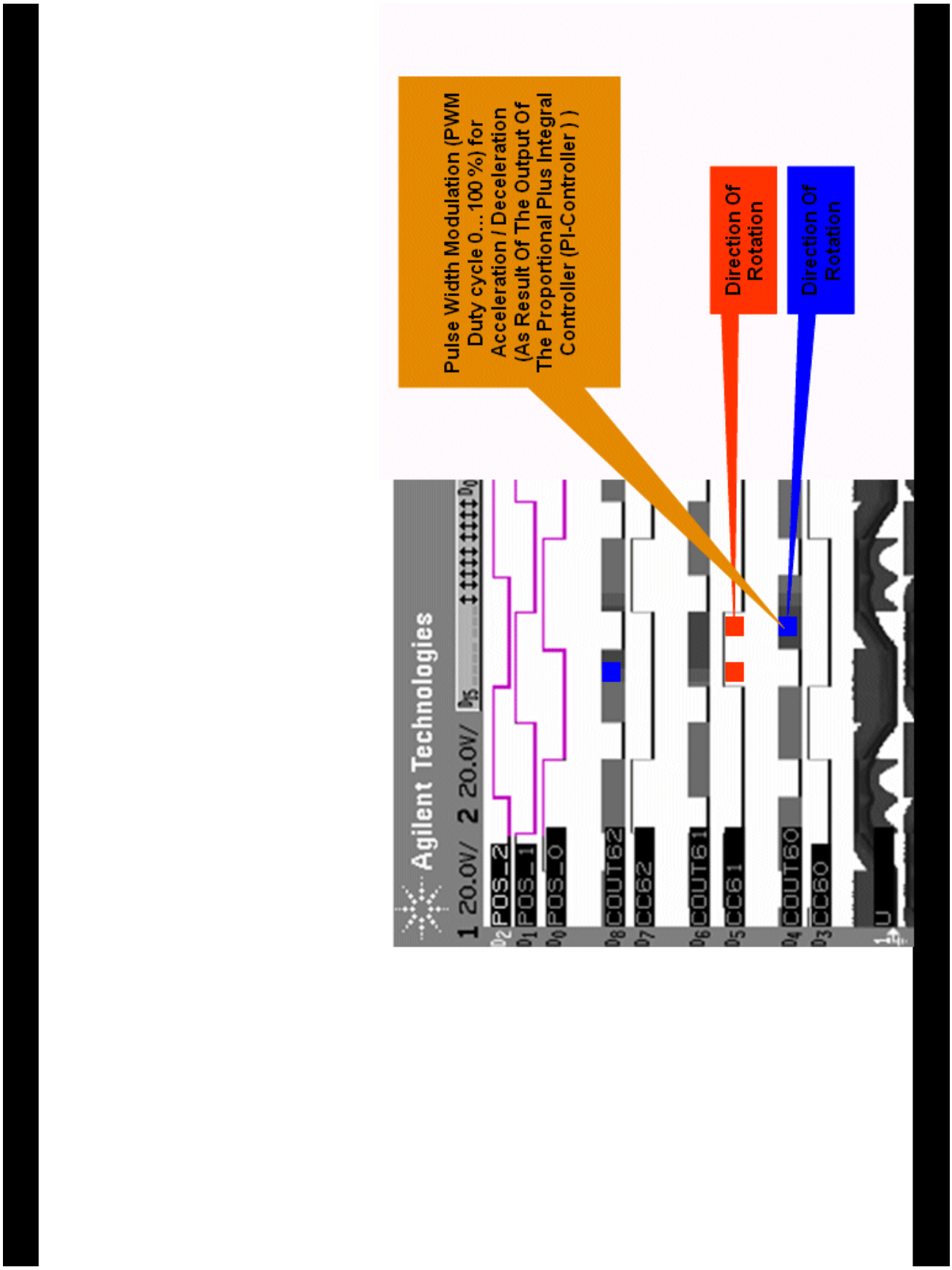
Operation of a Brushless DC Motor

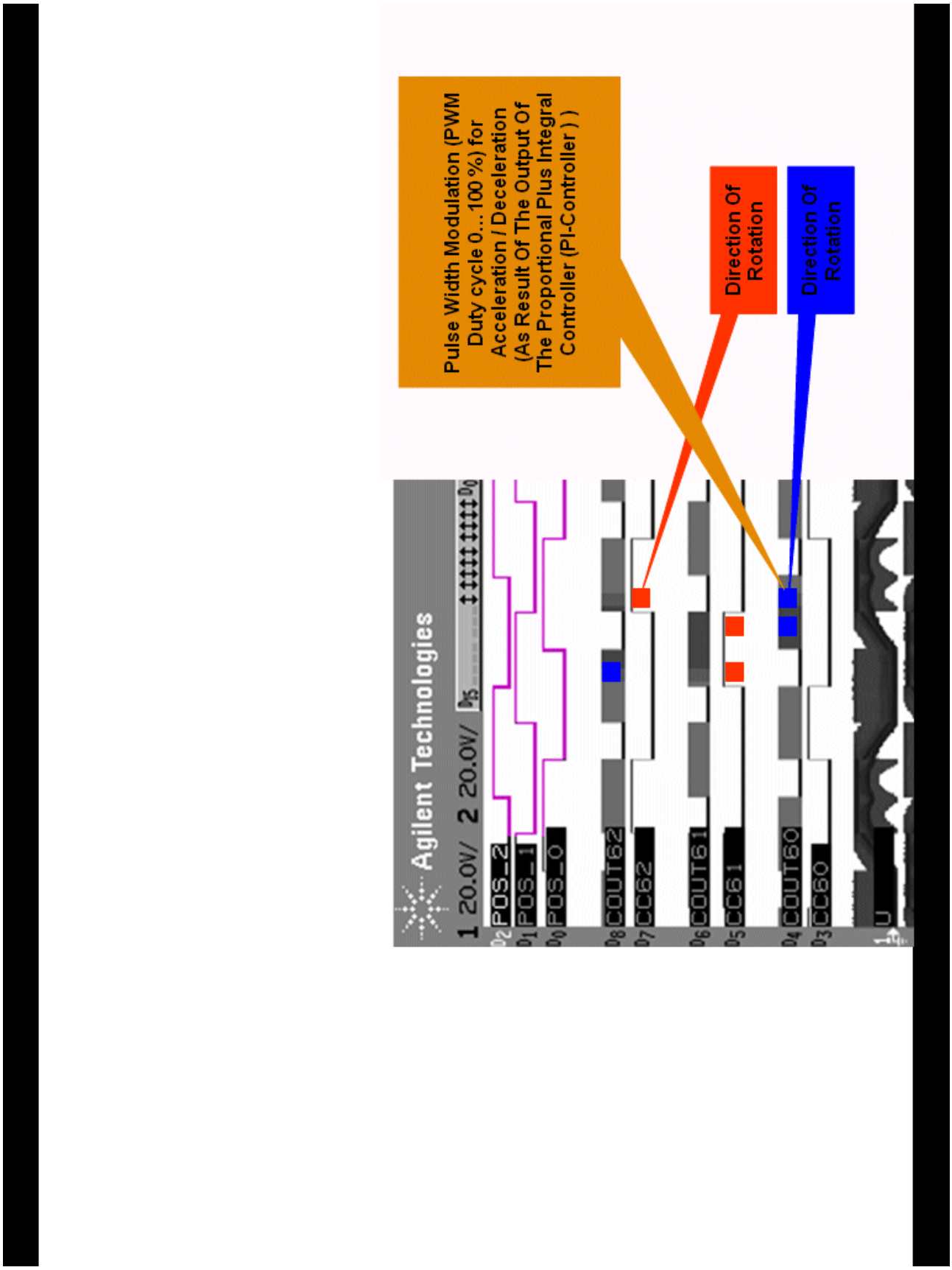
Only two of the three phases are energized in each state of the B6 full-bridge.

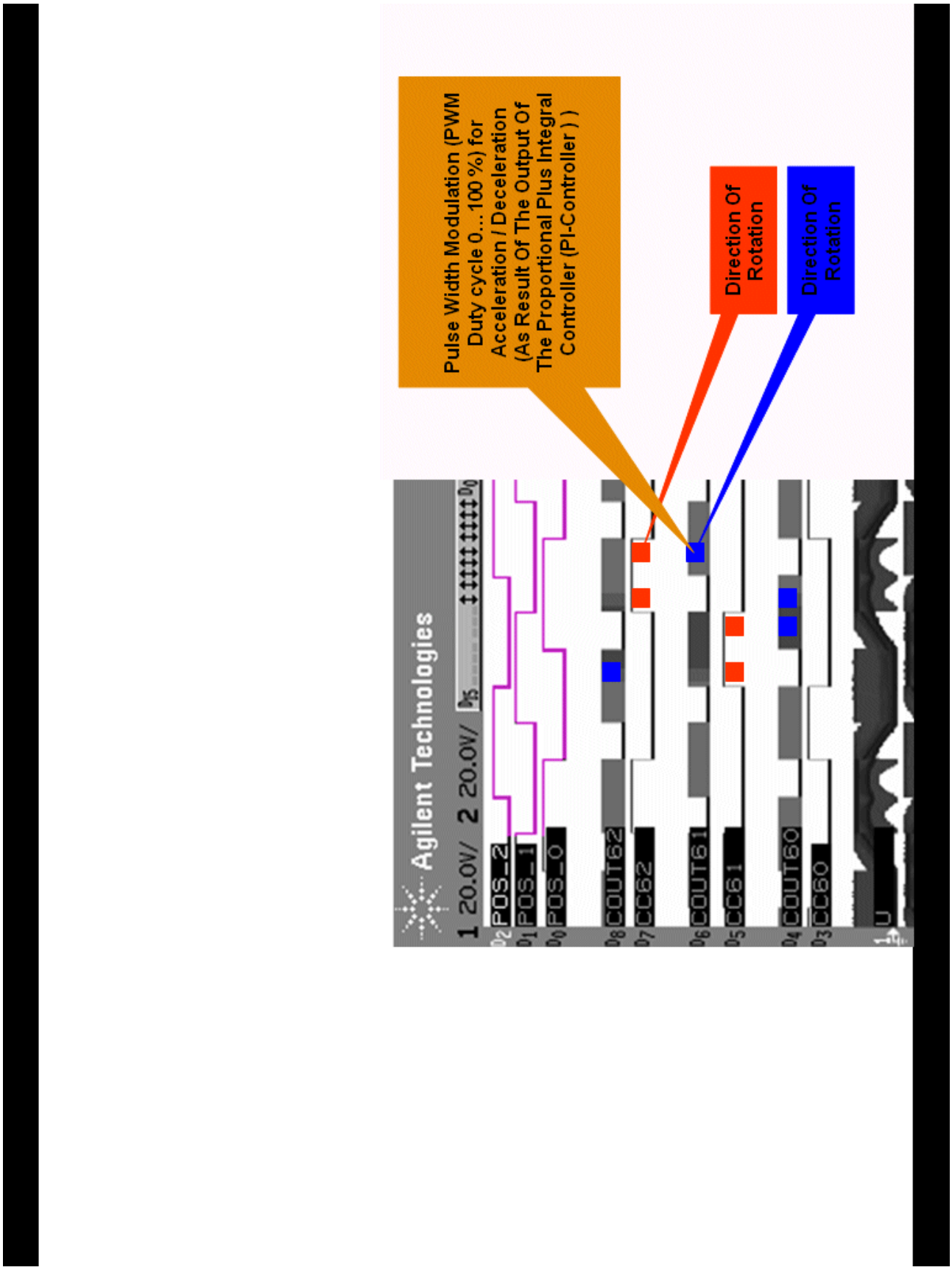
Electronical Commutation (No Mechanical Commutation)

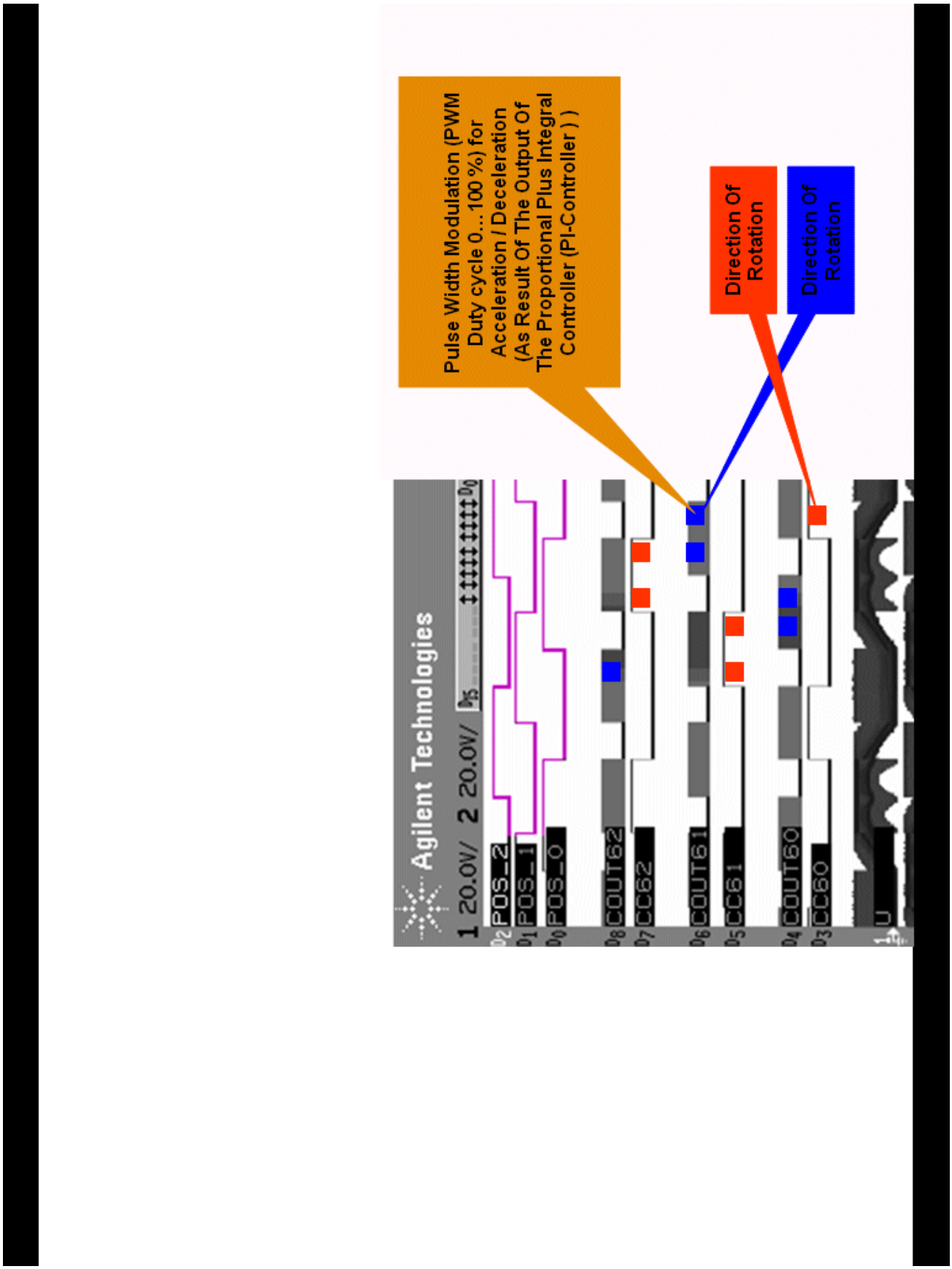
The Motor Position Is Reported By Three In-Built Hall Sensors. The Hall Sensors Arranged Offset By 120° Provide Six Different Switch Combinations Per Revolution. The Three Partial Windings Are Now Supplied In Six Different Conducting Phases In Accordance With The Sensor Information. The Current And Voltage Curves Are Block-Shaped.

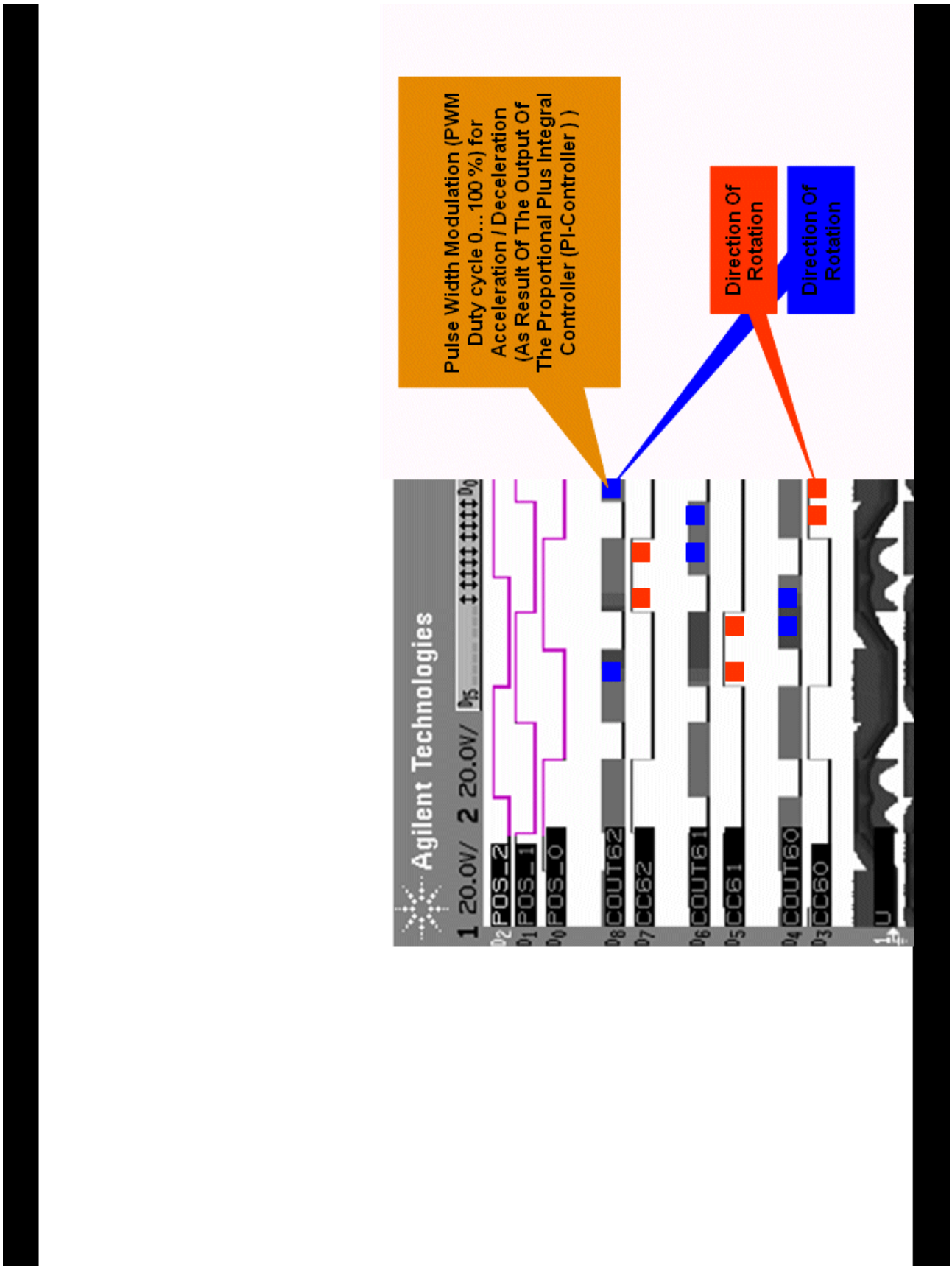

Motor Position (Hall Sensor Signals)
Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Tech
1 20.0V/ 2 20.
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

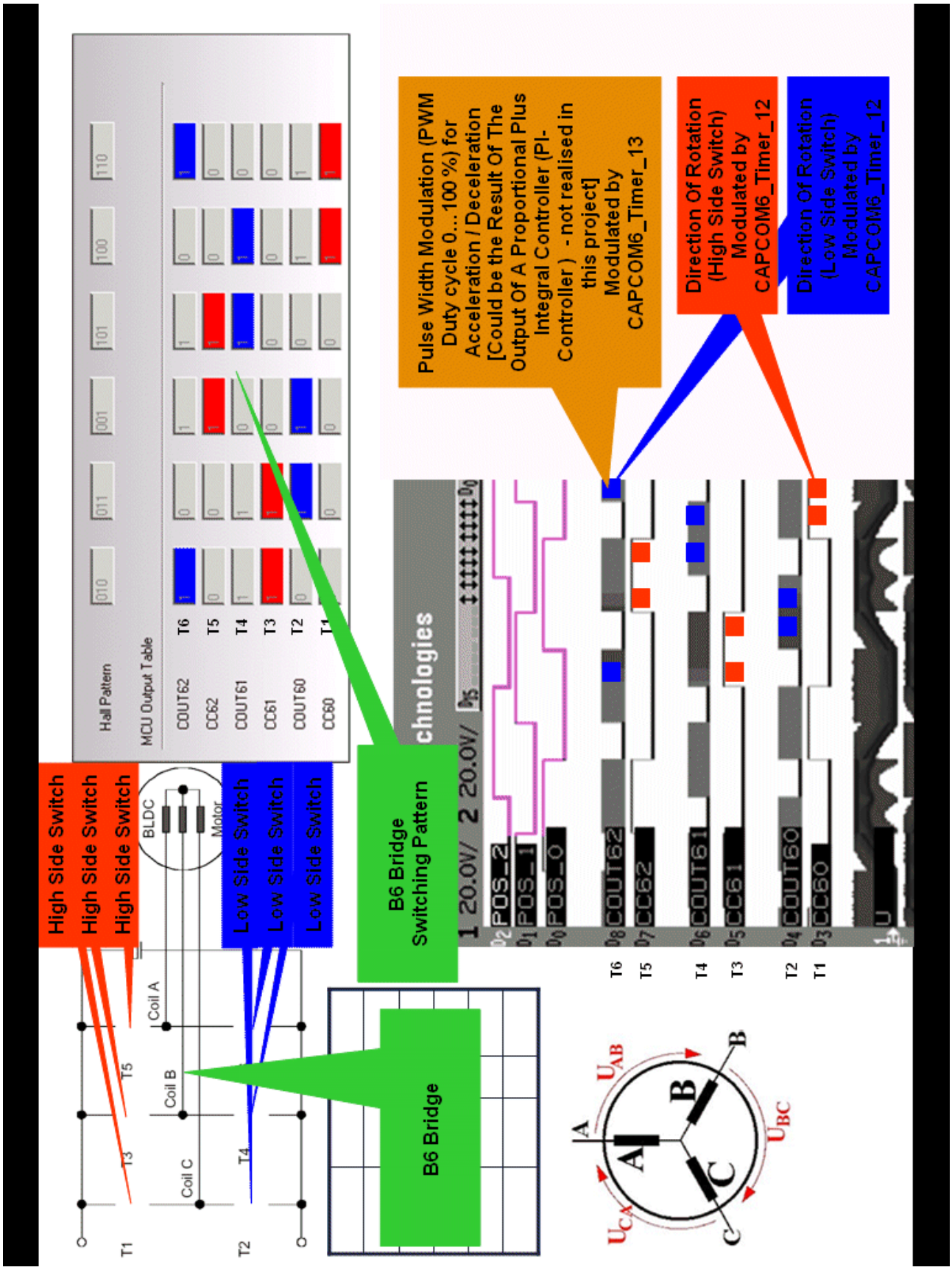

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Technologies
1 20.0V/ 2 20.0V/
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

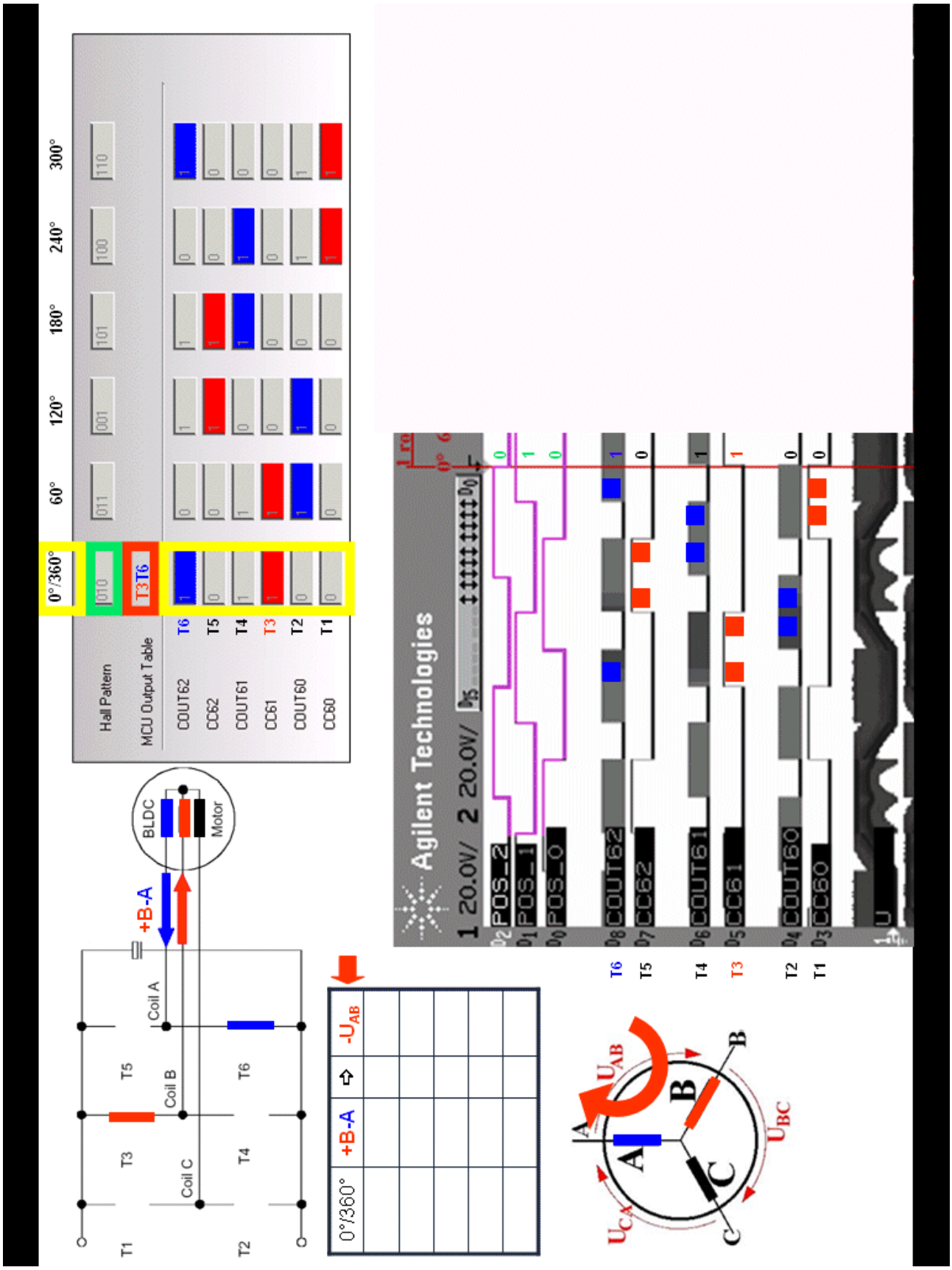

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Technologies
1 20.0V/ 2 20.0V/
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

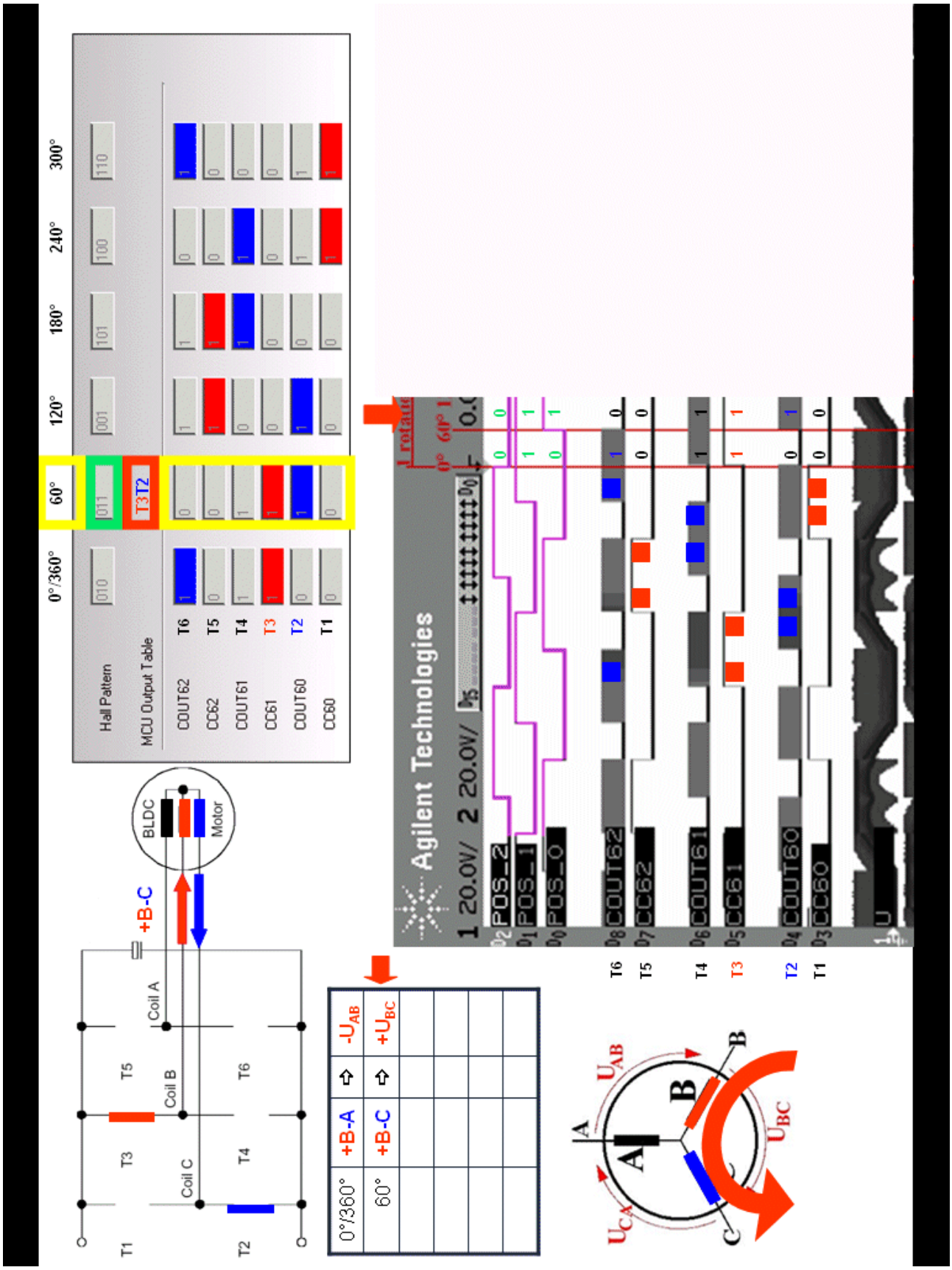

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Technologies
1 20.0V/ 2 20.0V/
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

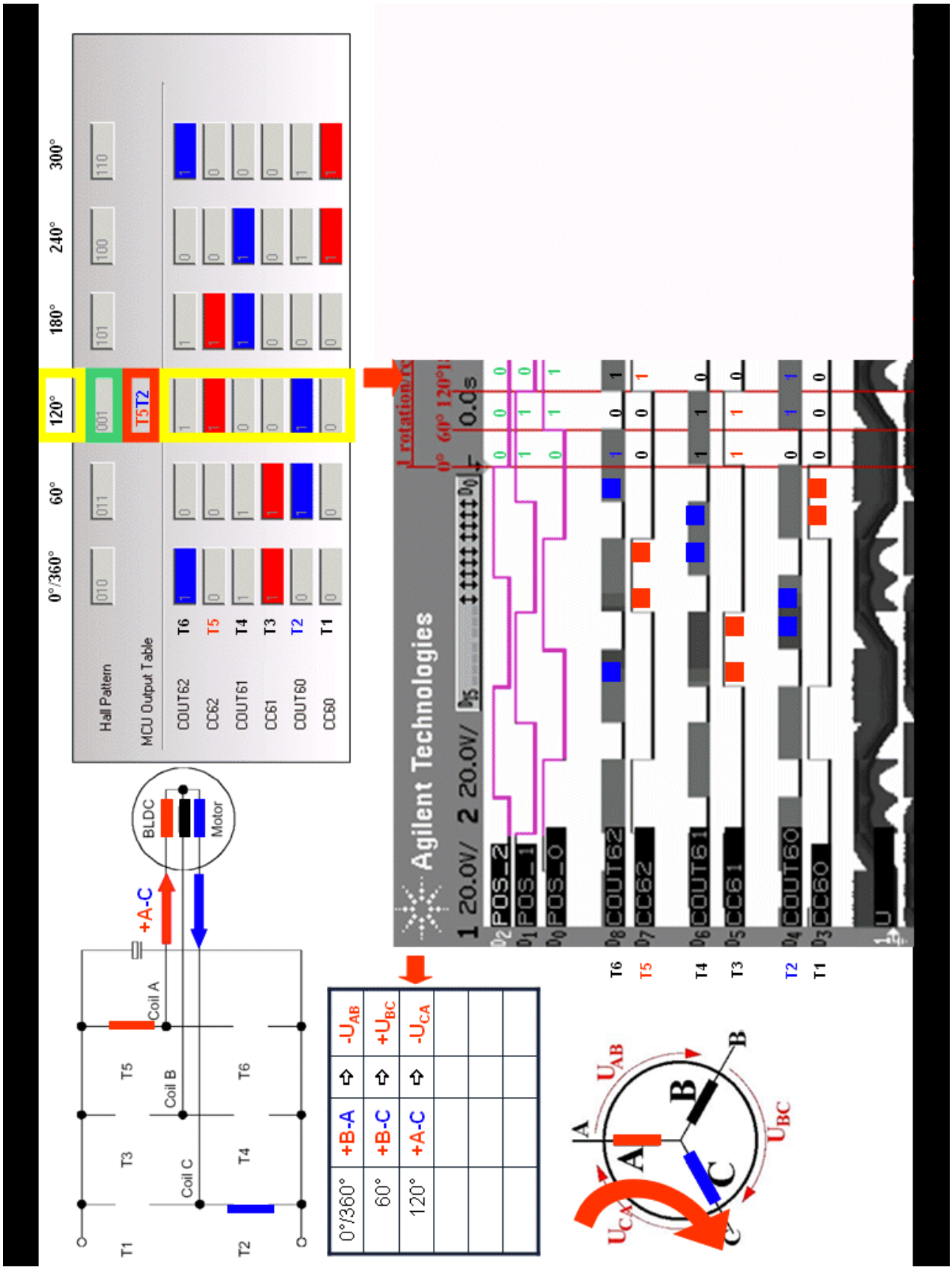

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Technologies
1 20.0V/ 2 20.0V/
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

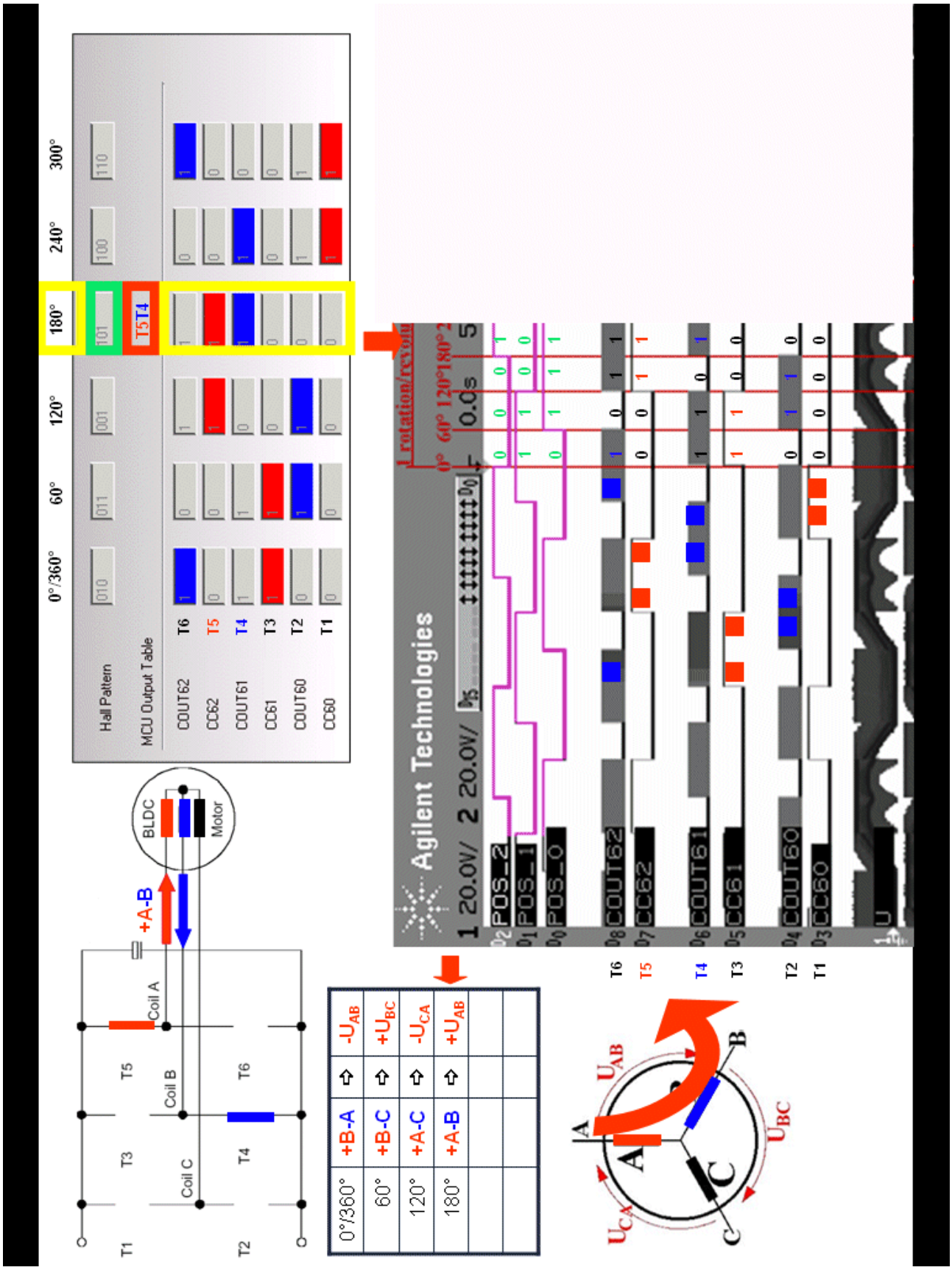

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration (As Result Of The Output Of The Proportional Plus Integral Controller (PI-Controller))
Direction Of Rotation
Direction Of Rotation
Agilent Technologies
1 20.0V/ 2 20.0V/
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U

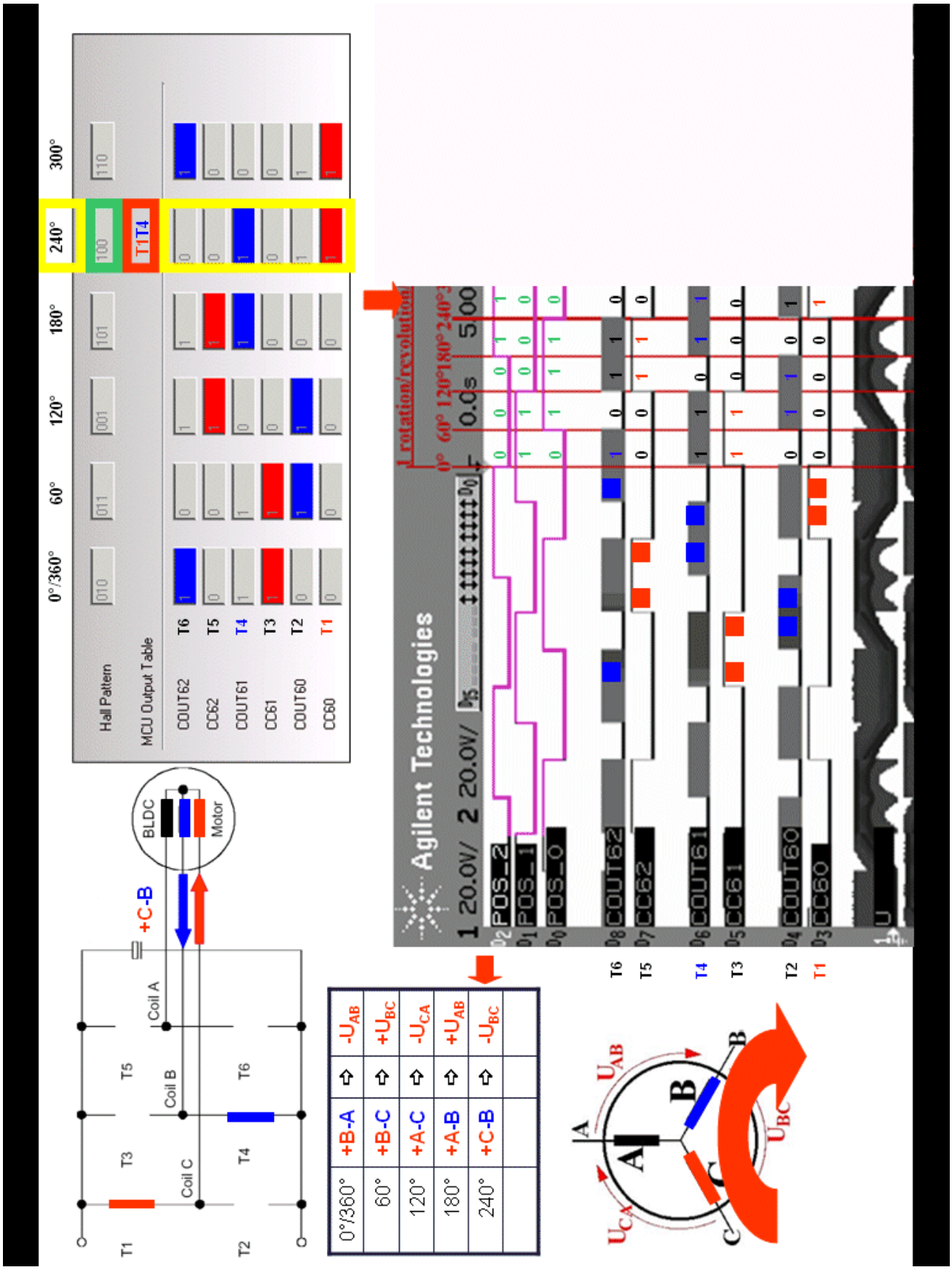

Pulse Width Modulation (PWM Duty cycle 0...100 %) for Acceleration / Deceleration [Could be the Result Of The Output Of A Proportional Plus Integral Controller (PI-Controller) - not realised in this project] Modulated by CAPCOM6_Timer_13
Direction Of Rotation (High Side Switch) Modulated by CAPCOM6_Timer_12
Direction Of Rotation (Low Side Switch) Modulated by CAPCOM6_Timer_12
B6 Bridge Switching Pattern
High Side Switch
High Side Switch
High Side Switch
Low Side Switch
Low Side Switch
Low Side Switch
B6 Bridge
BLDC Motor
Coil A
Coil B
Coil C
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
POS_2
POS_1
POS_0
U
UAB
UBC
UCA

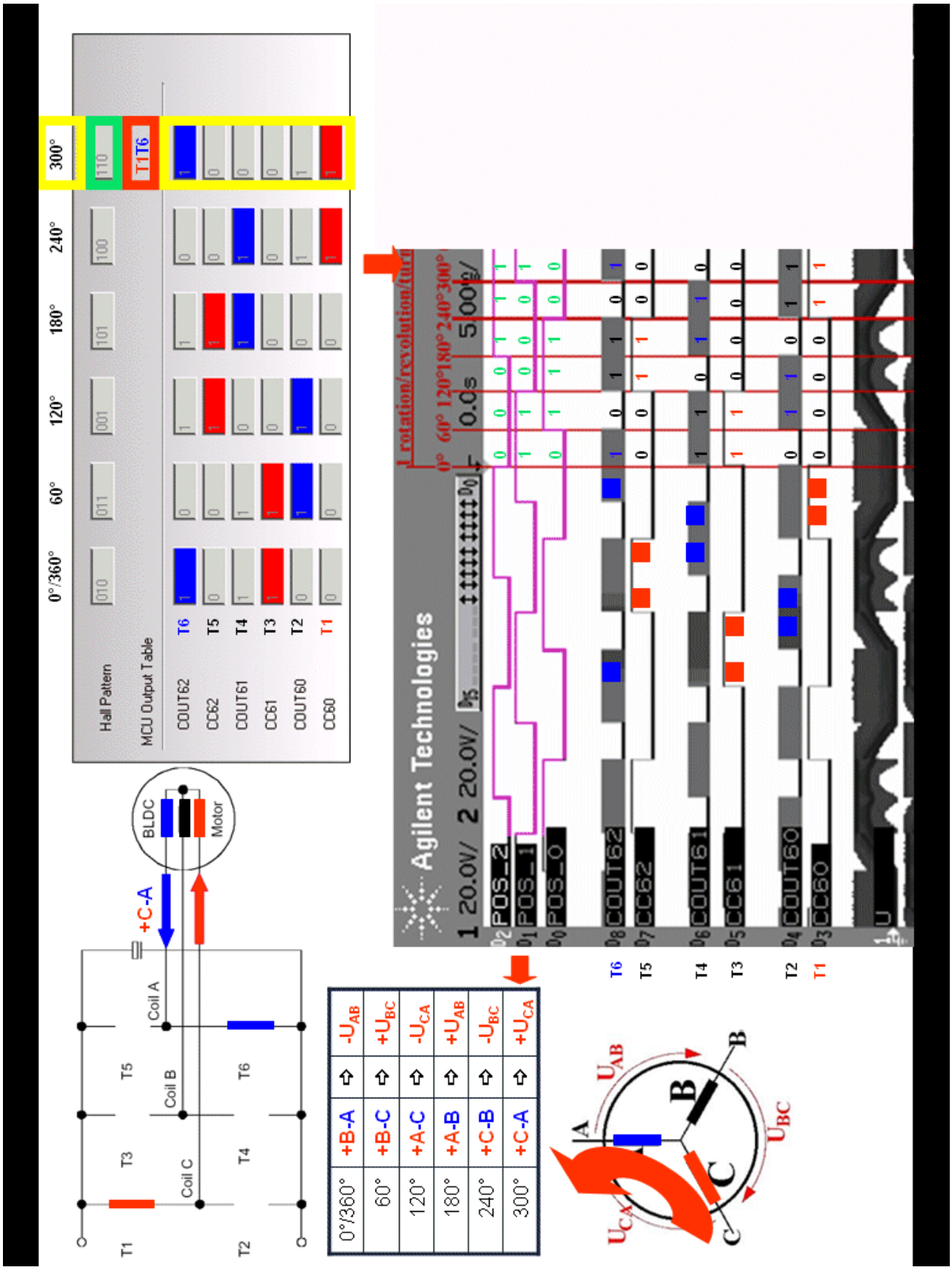

Hall Pattern
MCU Output Table
0°/360°
T3T6
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
BLDC
Motor
+B-A
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
0°/360°
+B-A
-U_AB
U_AB
U_BC
U_CA
A
B
C

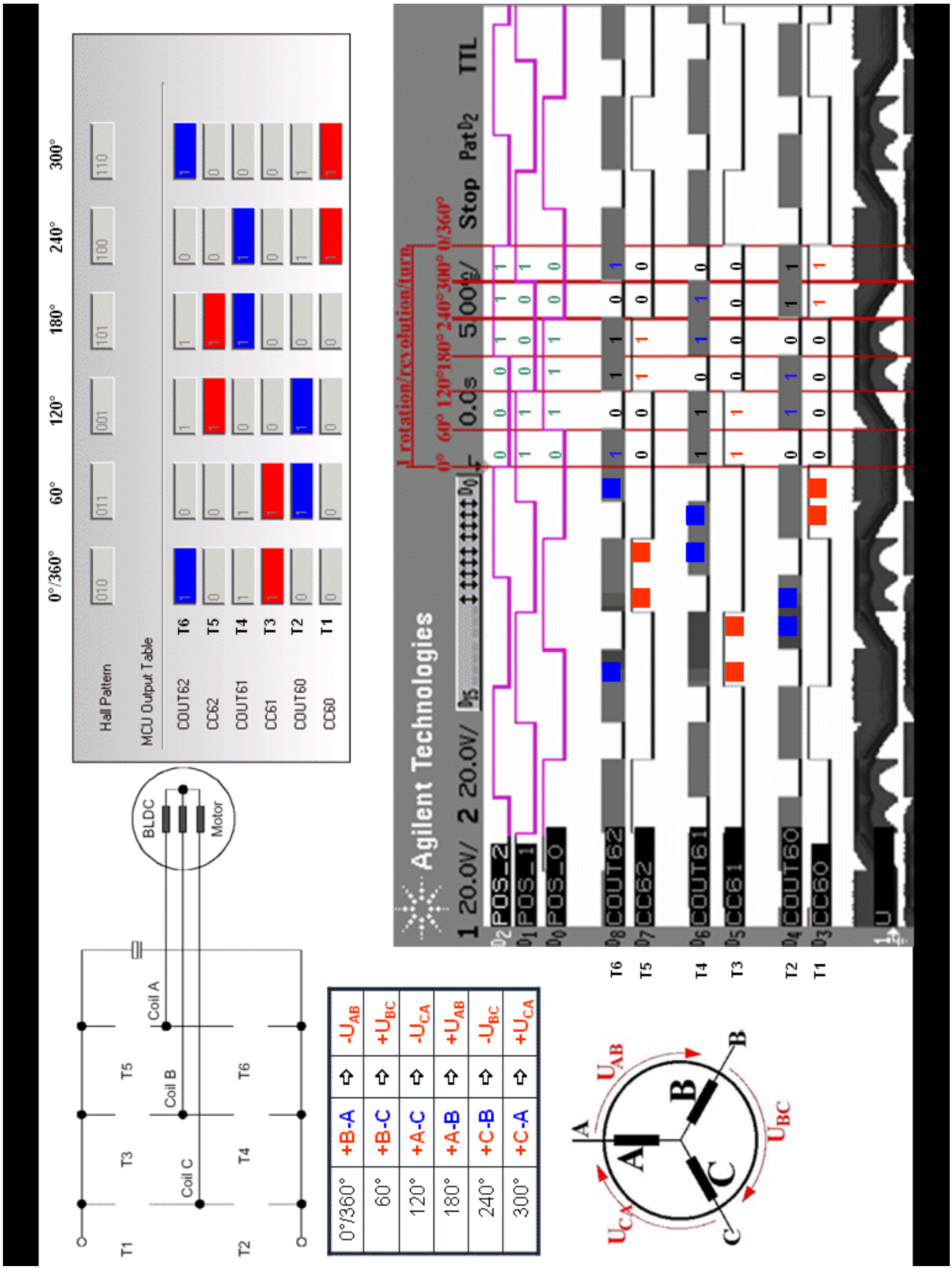

0°/360°
60°
120°
180°
240°
300°
Hall Pattern
MCU Output Table
T3T2
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
BLDC Motor
+B-C
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
0°/360°
60°
+B-A
+B-C
-UAB
+UBC
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
UAB
UBC
UCA

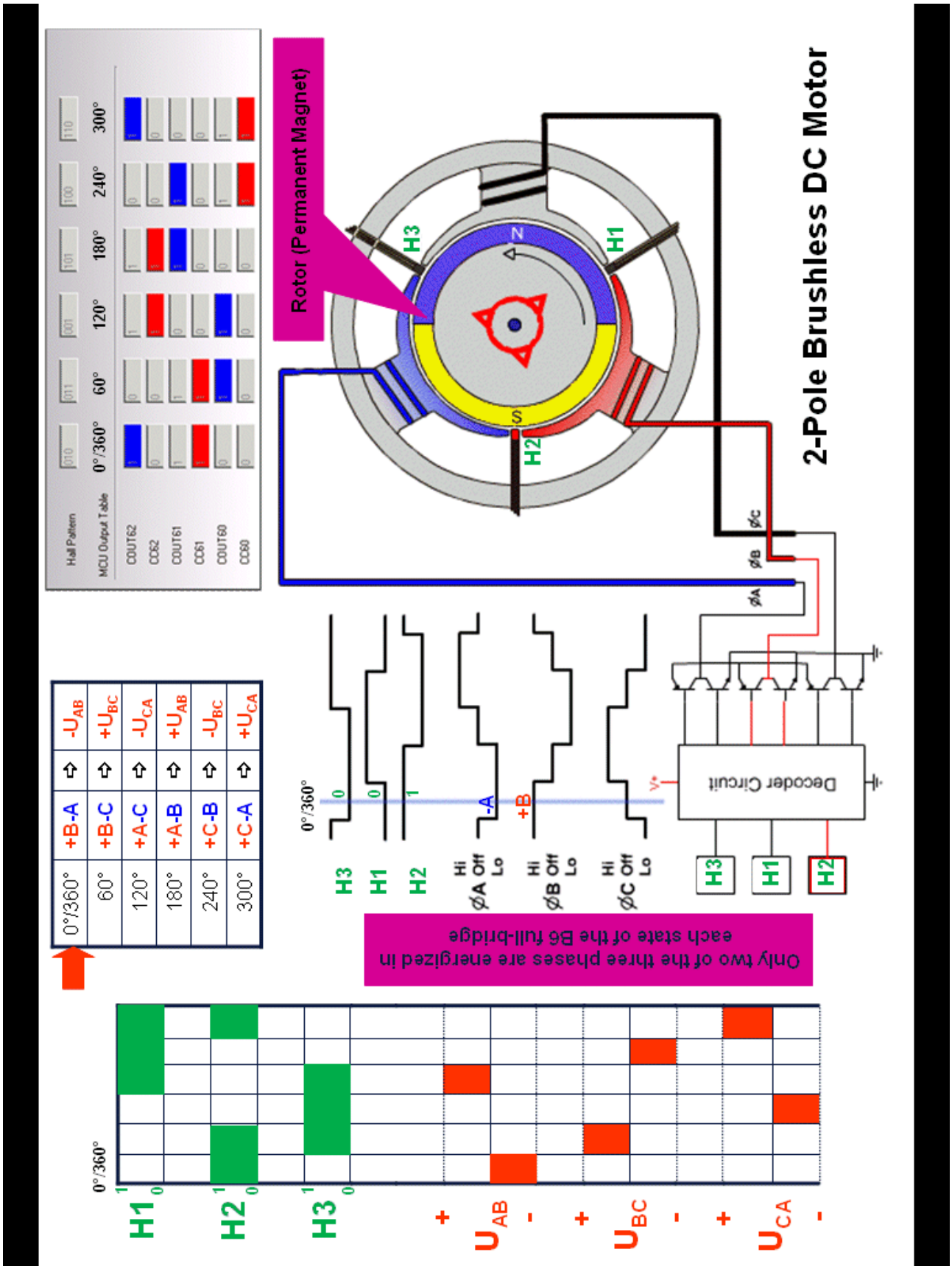

0°/360°
60°
120°
180°
240°
300°
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
T5T2
BLDC Motor
+A-C
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
-U_AB
+U_BC
-U_CA
+B-A
+B-C
+A-C
U_AB
U_BC
U_CA
A
B
C

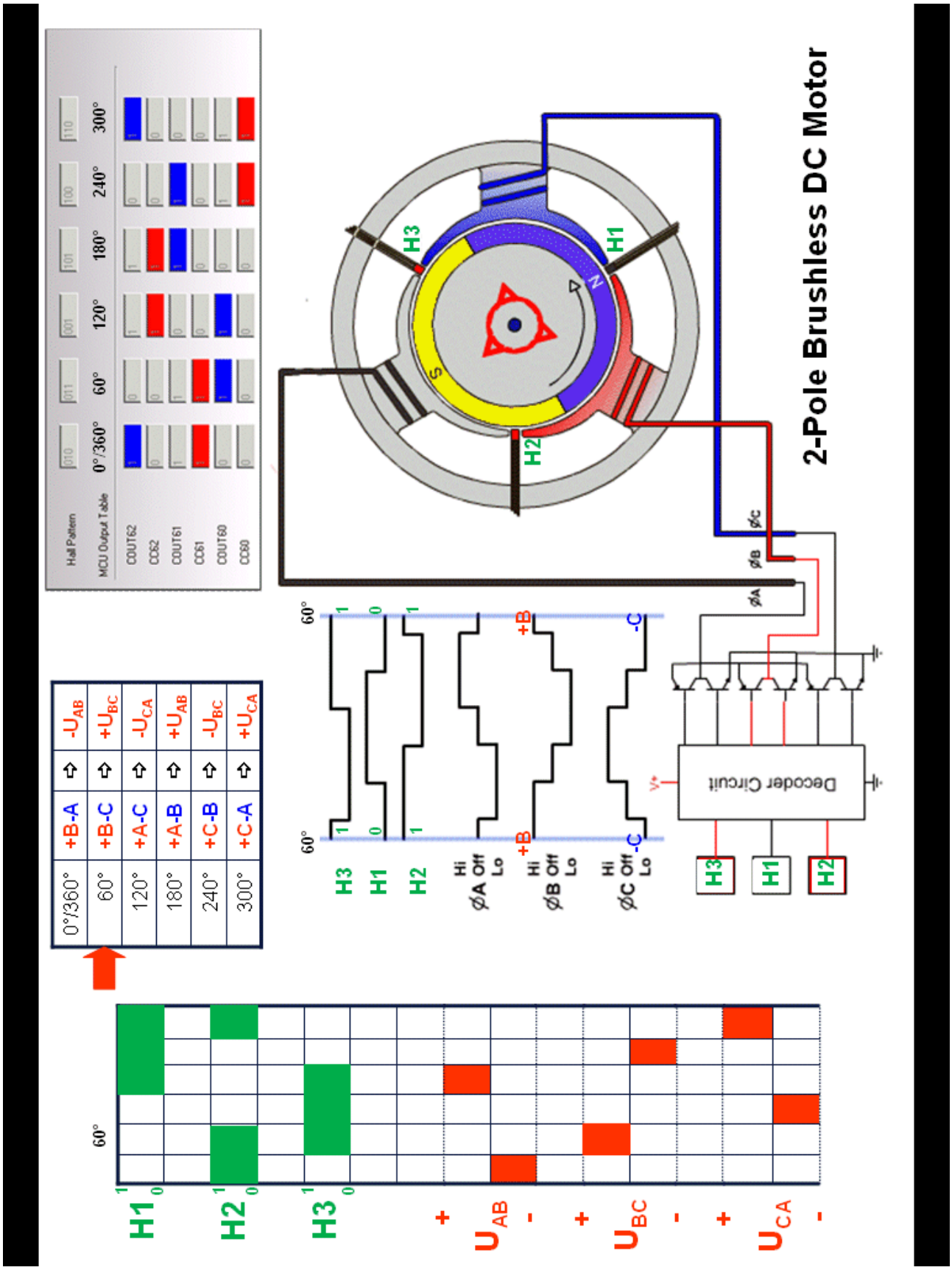

0°/360°
60°
120°
180°
240°
300°
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
U
BLDC
Motor
+A-B
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
0°/360° +B-A -U_AB
60° +B-C +U_BC
120° +A-C -U_CA
180° +A-B +U_AB
U_AB
U_BC
U_CA
A
B
C

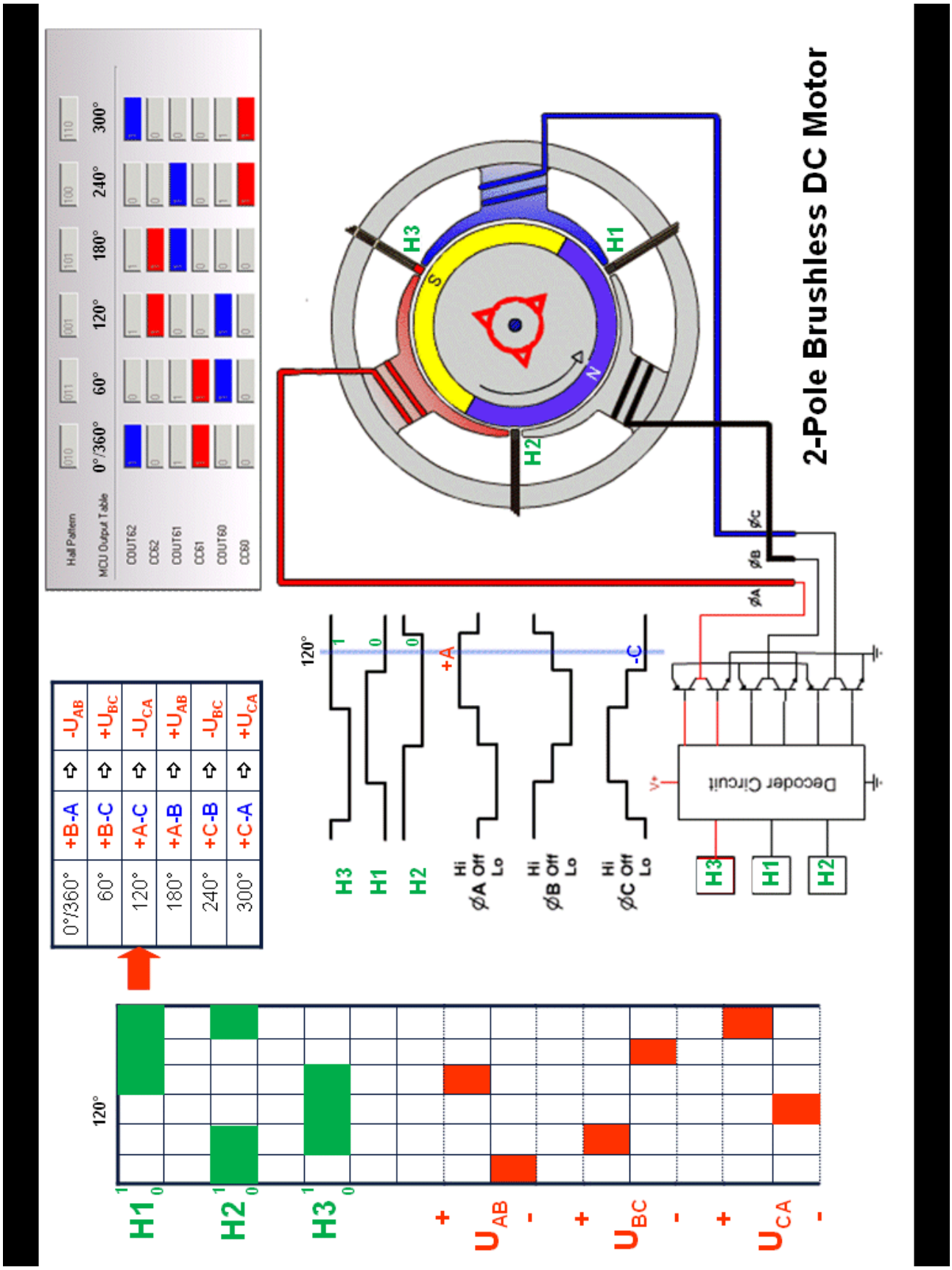

0°/360°
60°
120°
180°
240°
300°
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
T1T4
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
BLDC
Motor
+C-B
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
+B-A
+B-C
+A-C
+A-B
+C-B
-U_AB
+U_BC
-U_CA
+U_AB
-U_BC
U_AB
U_BC
U_CA
A
B
C

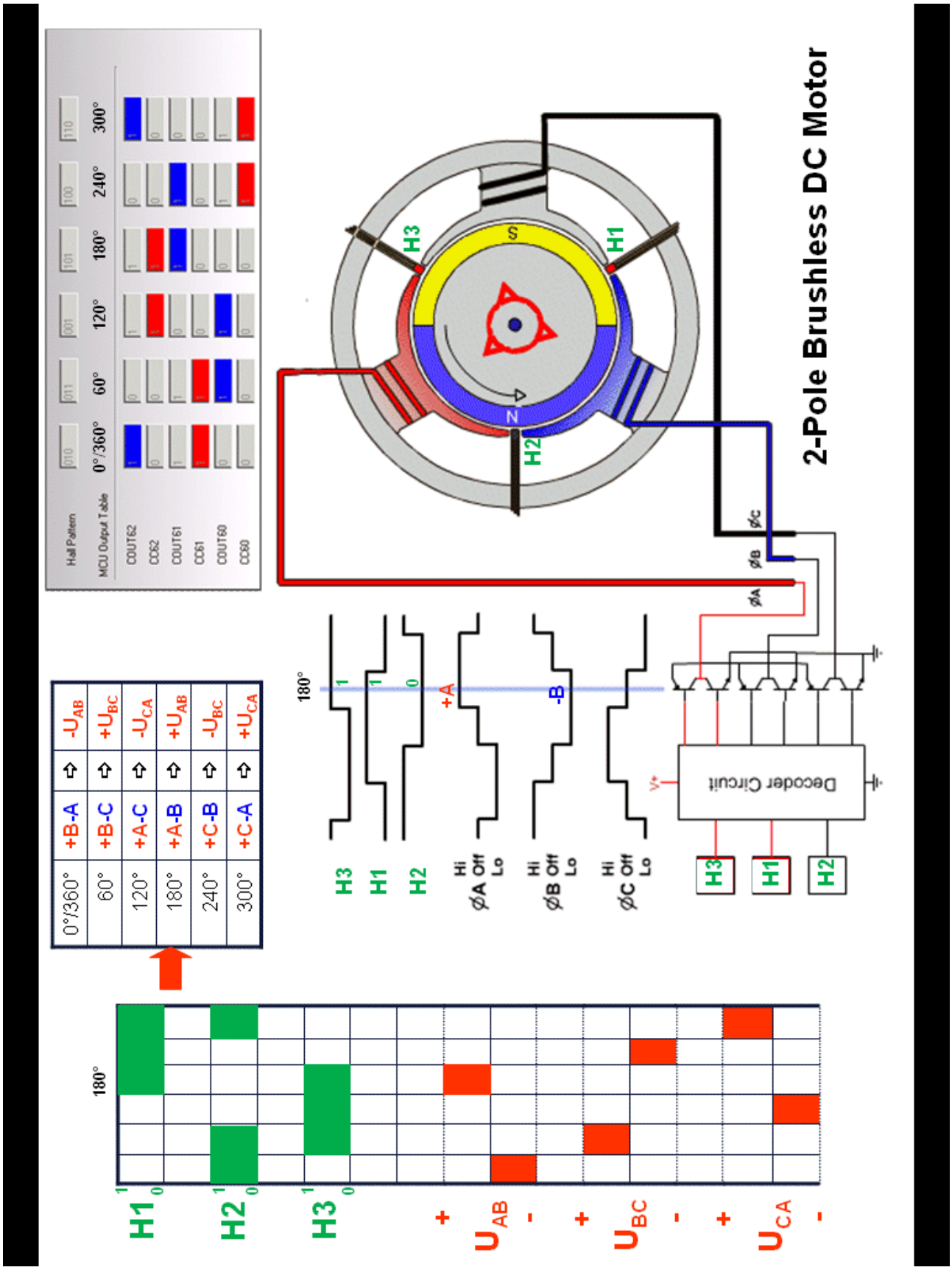

Hall Pattern
MCU Output Table
0°/360°
60°
120°
180°
240°
300°
010
011
001
101
100
110
T1T6
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
Agilent Technologies
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
BLDC
Motor
+C-A
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
0°/360° +B-A ⇧ -UAB
60° +B-C ⇧ +UBC
120° +A-C ⇧ -UCA
180° +A-B ⇧ +UAB
240° +C-B ⇧ -UBC
300° +C-A ⇧ +UCA
UAB
UBC
UCA
A
B
C

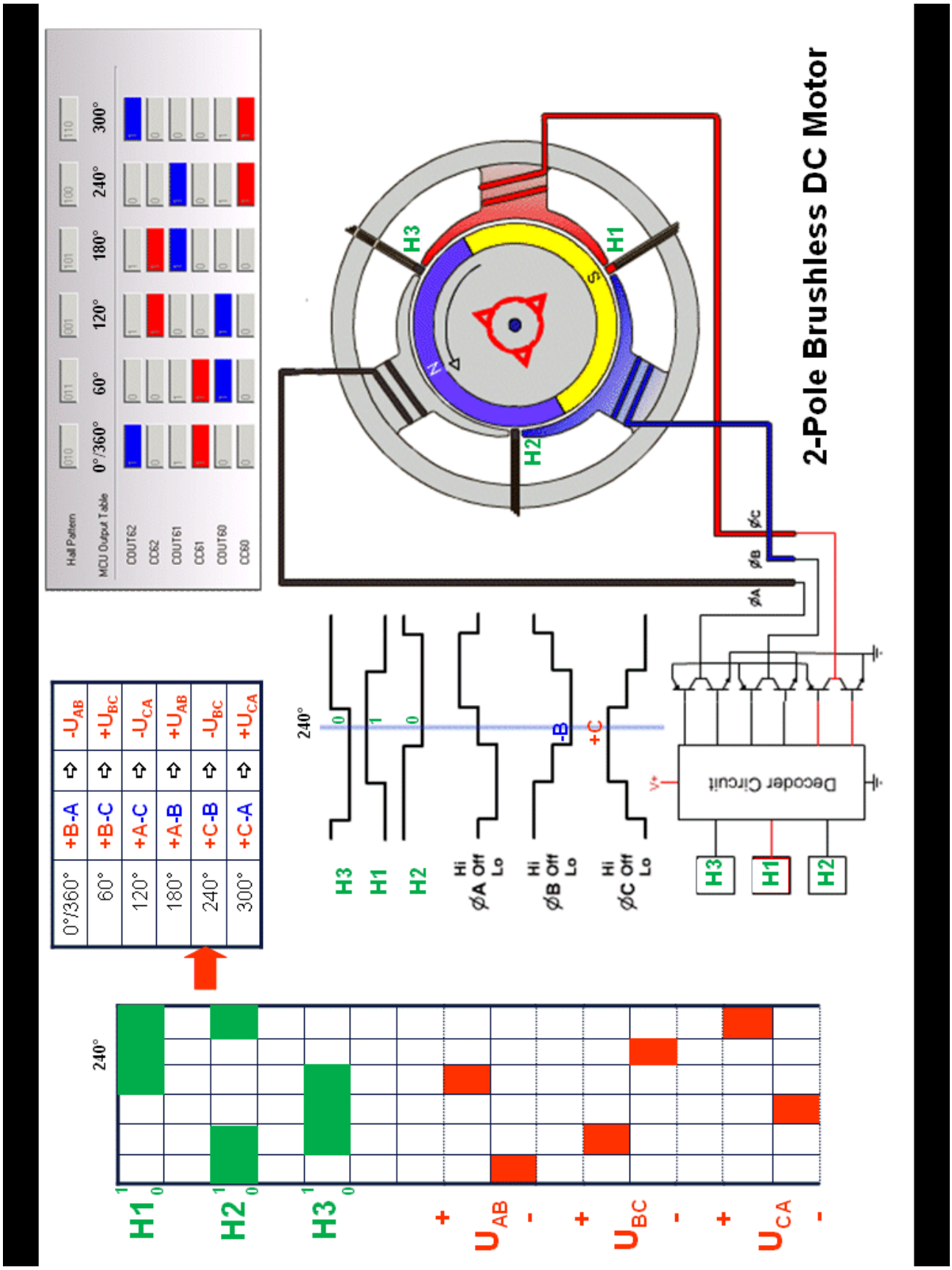

0°/360°
60°
120°
180°
240°
300°
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
T6
T5
T4
T3
T2
T1
Agilent Technologies
1 rotation/revolution/turn
POS_2
POS_1
POS_0
COUT62
CC62
COUT61
CC61
COUT60
CC60
Stop
Pat
TTL
BLDC Motor
Coil A
Coil B
Coil C
T1
T2
T3
T4
T5
T6
A
B
C
U_{AB}
U_{BC}
U_{CA}
0°/360° +B-A -U_{AB}
60° +B-C +U_{BC}
120° +A-C -U_{CA}
180° +A-B +U_{AB}
240° +C-B -U_{BC}
300° +C-A +U_{CA}

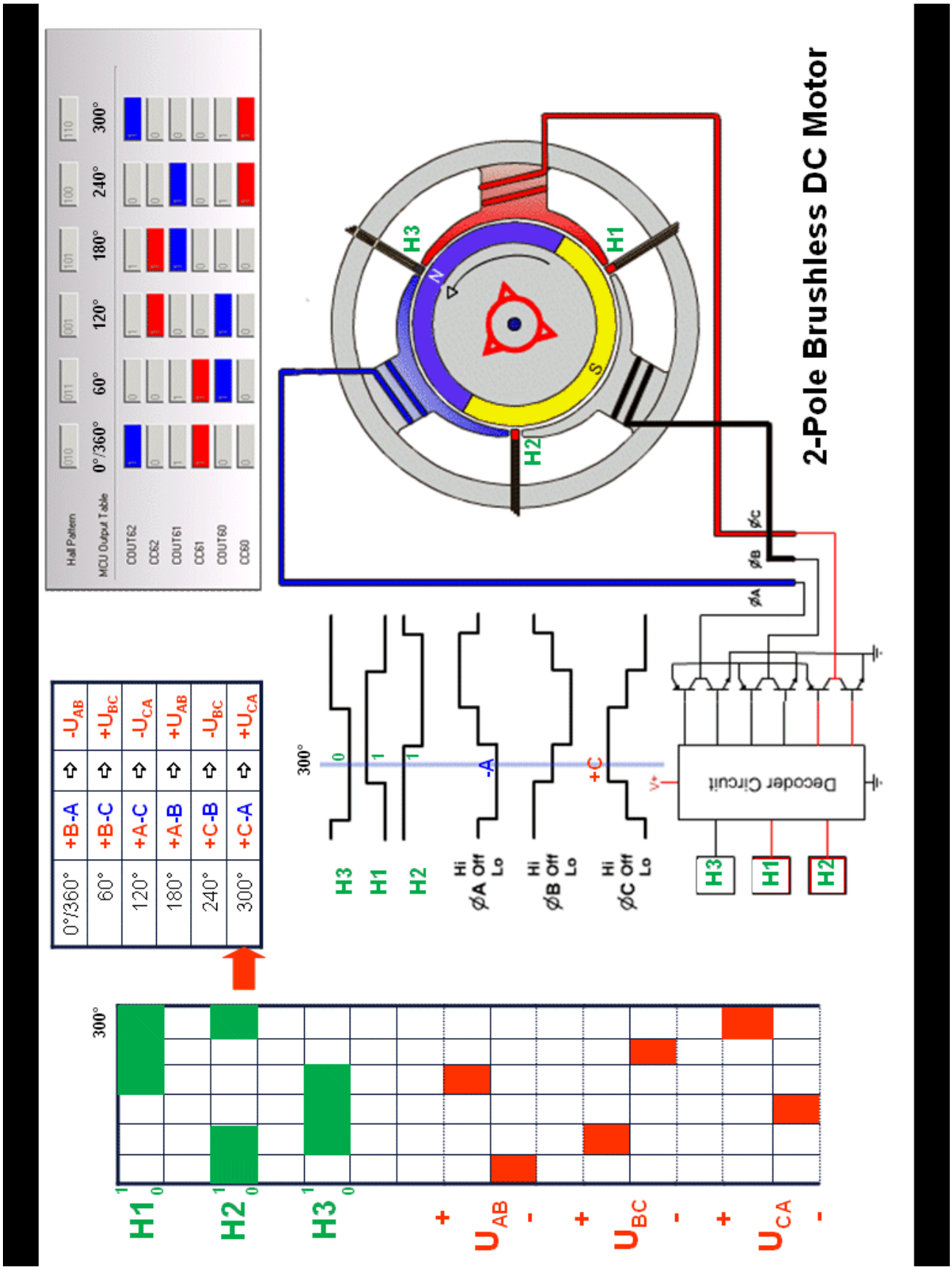

2-Pole Brushless DC Motor
Rotor (Permanent Magnet)
N
S
H1
H2
H3
ØA
ØB
ØC
Decoder Circuit
Only two of the three phases are energized in each state of the B6 full-bridge
0°/360°
60°
120°
180°
240°
300°
+B-A
+B-C
+A-C
+A-B
+C-B
+C-A
-UAB
+UBC
-UCA
+UAB
-UBC
+UCA
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
Hi
Off
Lo
-A
+B
UAB
UBC
UCA


2-Pole Brushless DC Motor
H1
H2
H3
N
S
ØA
ØB
ØC
Decoder Circuit
Hall Pattern
MCU Output Table
0°/360°
60°
120°
180°
240°
300°
COUT62
CC62
COUT61
CC61
COUT60
CC60
+B-A
+B-C
+A-C
+A-B
+C-B
+C-A
-UAB
+UBC
-UCA
+UAB
-UBC
+UCA
Hi
Off
Lo
60°
H1
H2
H3
UAB
UBC
UCA


2-Pole Brushless DC Motor
H1
H2
H3
S
N
ØA
ØB
ØC
Hall Pattern
MCU Output Table
0°/360°
60°
120°
180°
240°
300°
COUT62
CC62
COUT61
CC61
COUT60
CC60
0°/360° +B-A -UAB
60° +B-C +UBC
120° +A-C -UCA
180° +A-B +UAB
240° +C-B -UBC
300° +C-A +UCA
120°
+A
-C
Hi
Off
Lo
ØA
ØB
ØC
Decoder Circuit
V+
H1
H2
H3
UAB
UBC
UCA

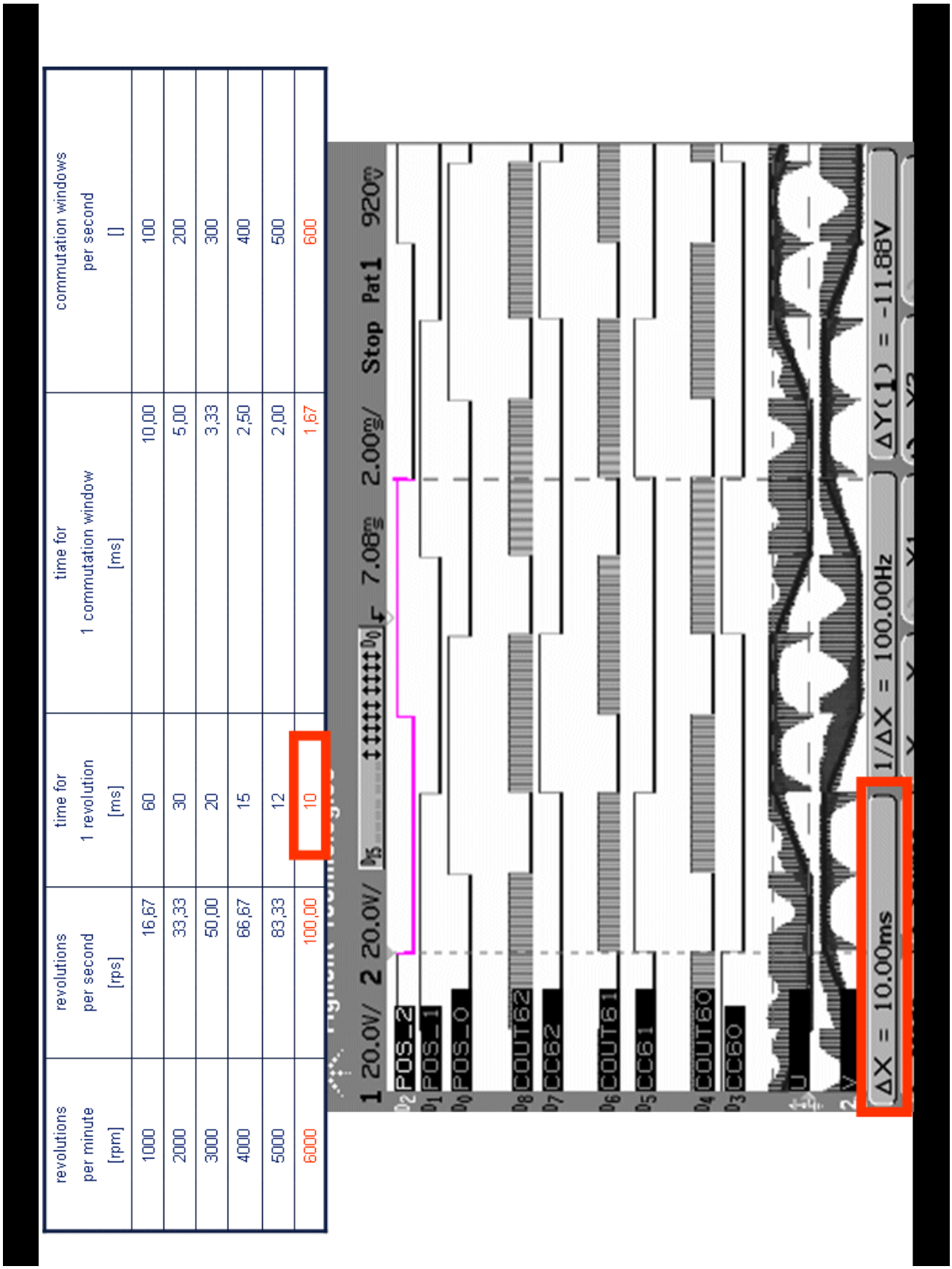

2-Pole Brushless DC Motor
H3
H1
H2
S
N
ØA
ØB
ØC
Decoder Circuit
180°
+A
-B
Hi
Off
Lo
ØA
ØB
ØC
0°/360°
60°
120°
180°
240°
300°
+B-A
+B-C
+A-C
+A-B
+C-B
+C-A
-UAB
+UBC
-UCA
+UAB
-UBC
+UCA
Hall Pattern
MCU Output Table
COUT62
CC62
COUT61
CC61
COUT60
CC60
UAB
UBC
UCA


2-Pole Brushless DC Motor
H1
H2
H3
S
N
ϕA
ϕB
ϕC
Decoder Circuit
H1
H2
H3
240°
H3
H1
H2
Hi
ϕA Off
Lo
Hi
ϕB Off
Lo
Hi
ϕC Off
Lo
-B
+C
Hall Pattern
MCU Output Table
0°/360°
60°
120°
180°
240°
300°
COUT62
CC62
COUT61
CC61
COUT60
CC60
0°/360° +B-A -UAB
60° +B-C +UBC
120° +A-C -UCA
180° +A-B +UAB
240° +C-B -UBC
300° +C-A +UCA
240°
H1
H2
H3
+ UAB -
+ UBC -
+ UCA -

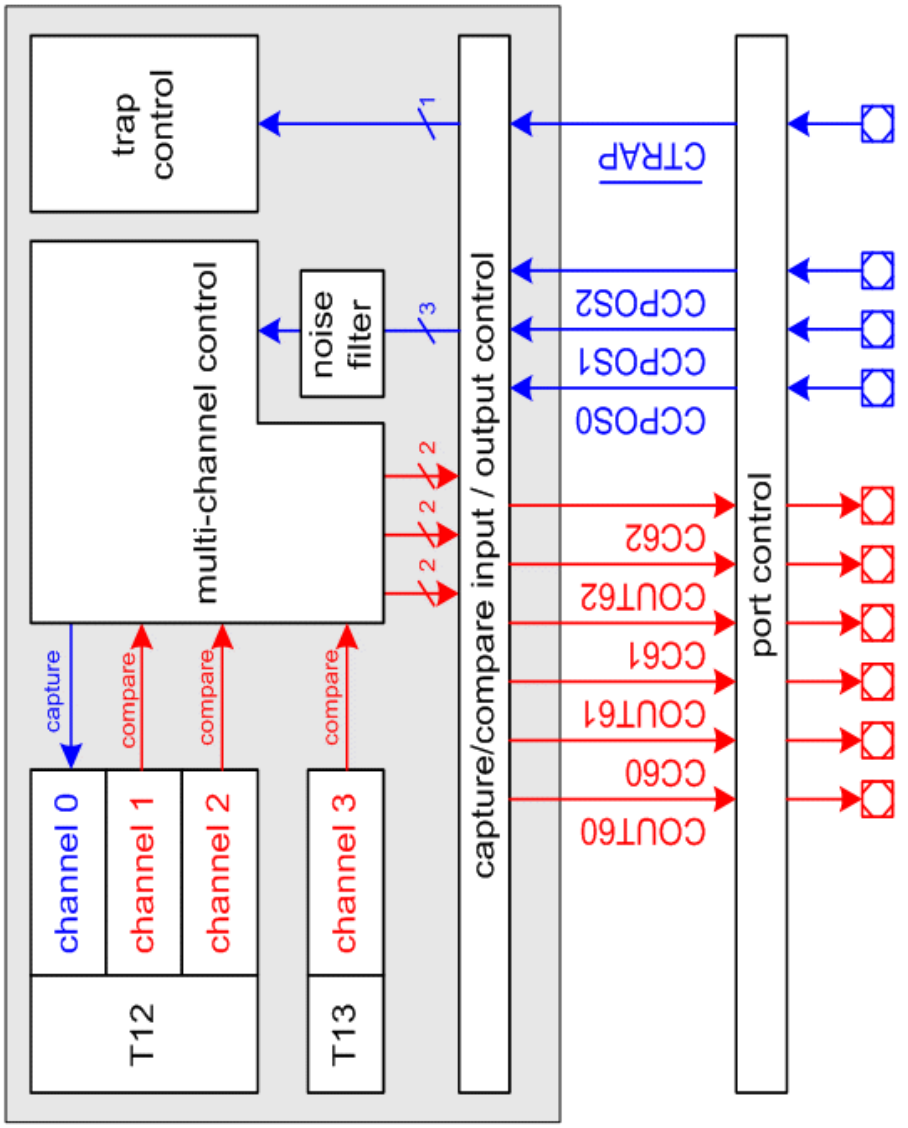

2-Pole Brushless DC Motor
H1
H2
H3
N
S
ØA
ØB
ØC
Decoder Circuit
Hall Pattern
MCU Output Table
0°/360°
60°
120°
180°
240°
300°
COUT62
CC62
COUT61
CC61
COUT60
CC60
+B-A
+B-C
+A-C
+A-B
+C-B
+C-A
-UAB
+UBC
-UCA
+UAB
-UBC
+UCA
300°
Hi
Off
Lo
-A
+C
H1
H2
H3
UAB
UBC
UCA

Time – Measurement:

revolutions per minute [rpm]	revolutions per second [rps]	time for 1 revolution [ms]	time for 1 commutation window [ms]	commutation windows per second []
1000	16,67	60	10,00	100
2000	33,33	30	5,00	200
3000	50,00	20	3,33	300
4000	66,67	15	2,50	400
5000	83,33	12	2,00	500
6000	100,00	10	1,67	600



Block Diagram CAPCOM6E for BLDC Usage



Dead-time Generation

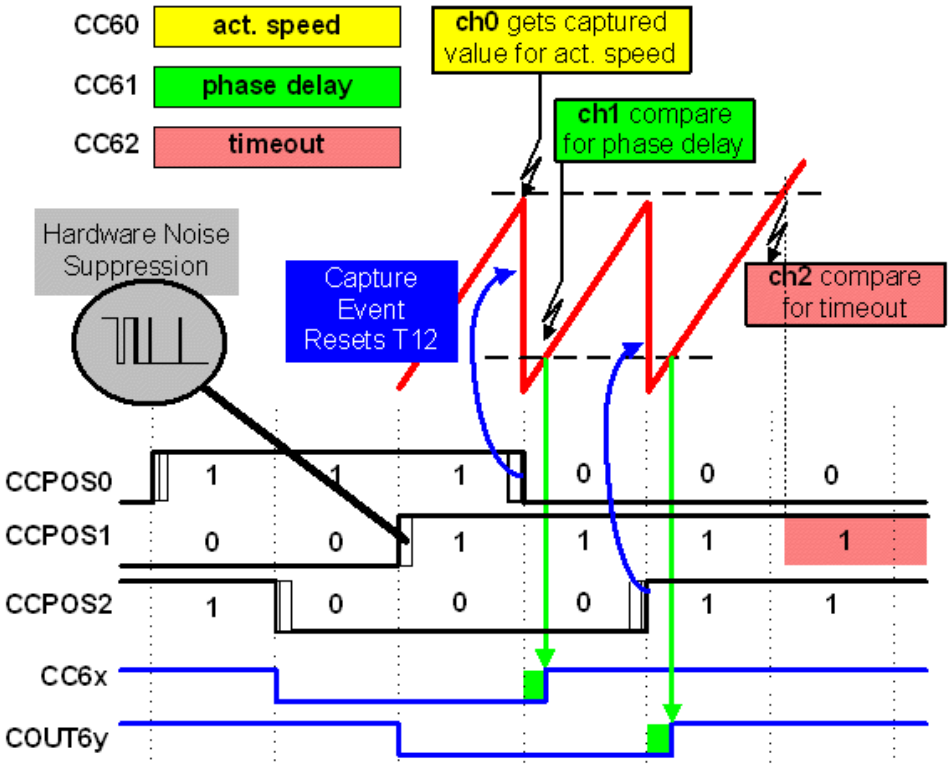
This capture/compare unit
contains a programmable dead-time counter delaying
the passive to active edge of the switching signals
(the active to passive edge is not delayed)

Hall Sensor Mode

In the Hall Sensor Mode,
timer T12 is used to measure the rotational speed of the motor
(channel 0 in capture mode)
and to control the phase delay before switching to the next state
(channel 1 in compare mode).

Use of the CAPCOM6E

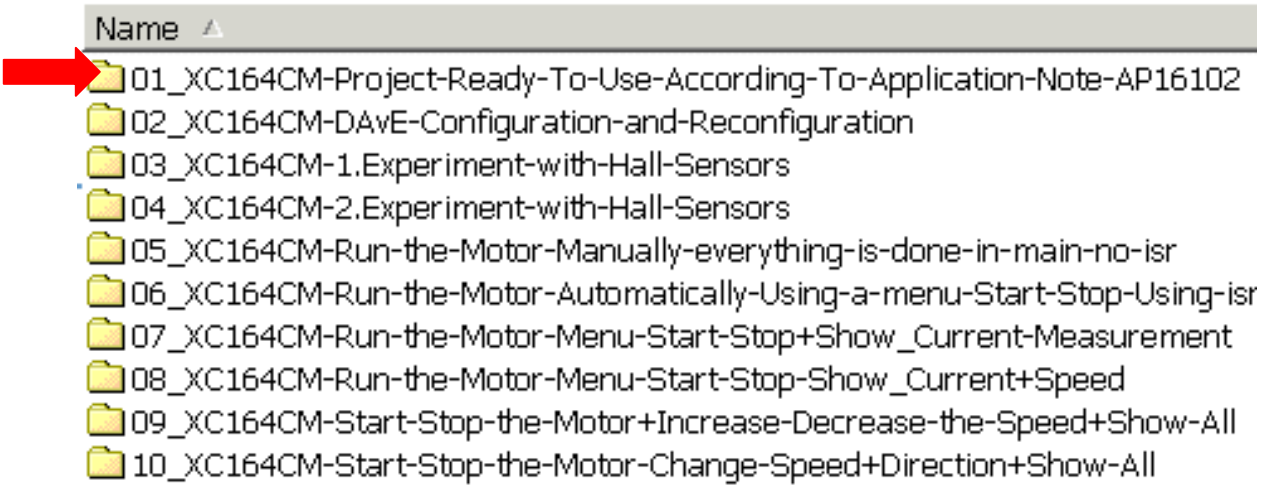
- HW-noise filter on CCPOSx inputs
- automatic reset of T12 with interrupt
- actual speed by capture ch0
- phase delay function on ch1
- time out function on ch2



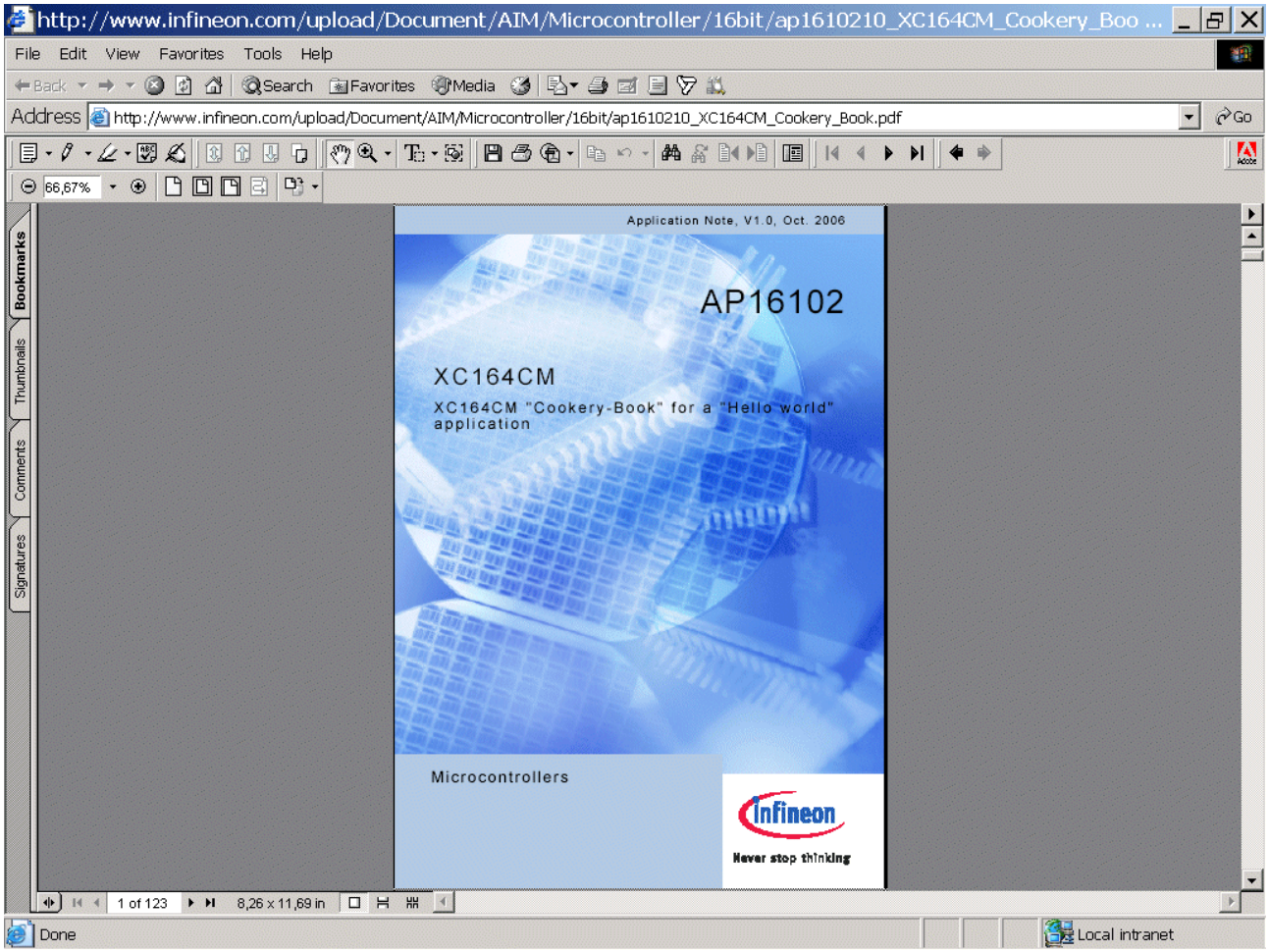
1.)

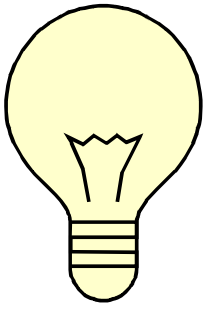
Using the "XC164CM Cookery Book" Step By Step:

Relevant Project:



It is necessary to follow all instructions in Cookery Book AP16102 (http://www.infineon.com/upload/Document/AIM/Microcontroller/16bit/ap1610210_XC164CM_Cookery_Book.pdf) step by step, as these will form the basis for the instructions which follow later.

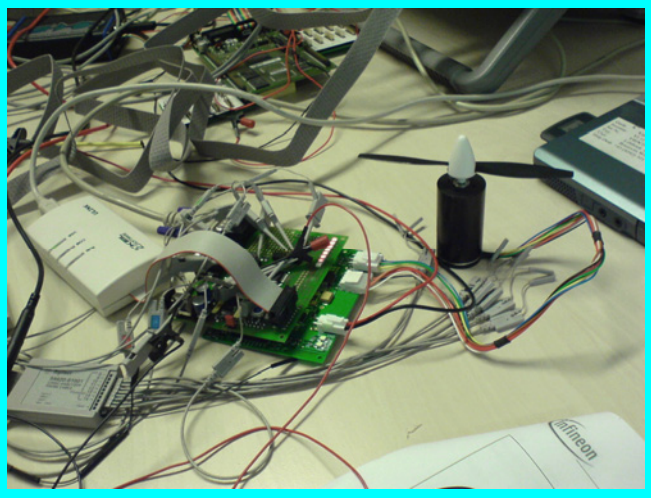




Note:
In the following steps of this document we will expand the “Hello World Application” (according to Application Note AP16102) with the requirements to drive the BLDC-Motor in Hall-Sensor-Mode.

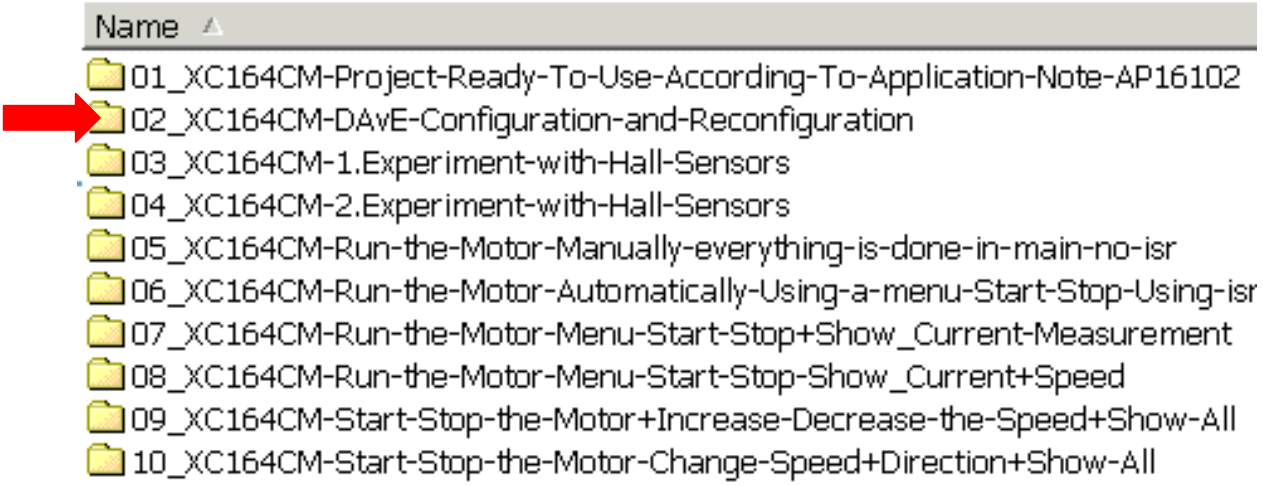
2.)

Let's Get Started:



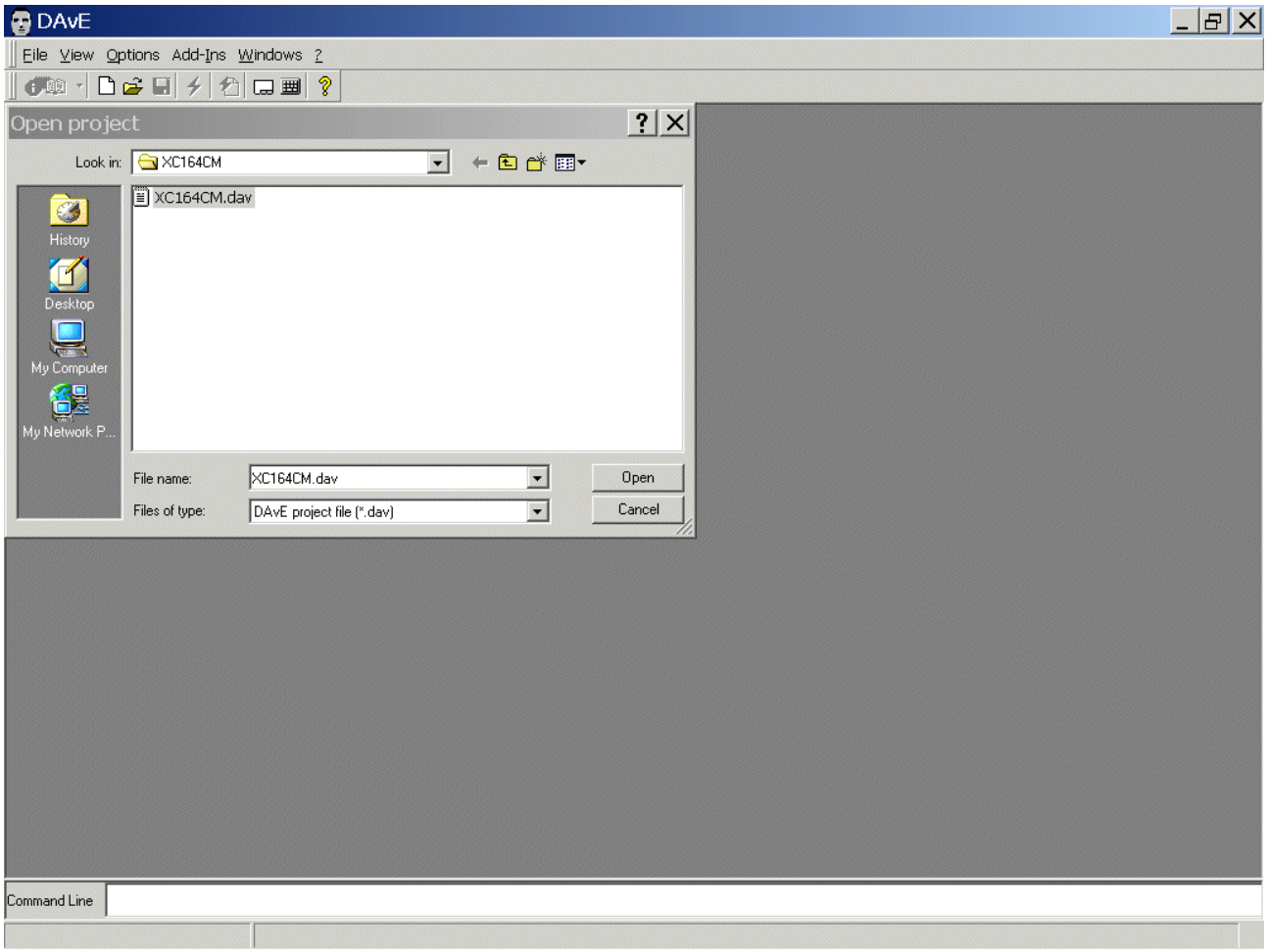
Configuring and Reconfiguring of the DAvE Project Settings:

Relevant Project:

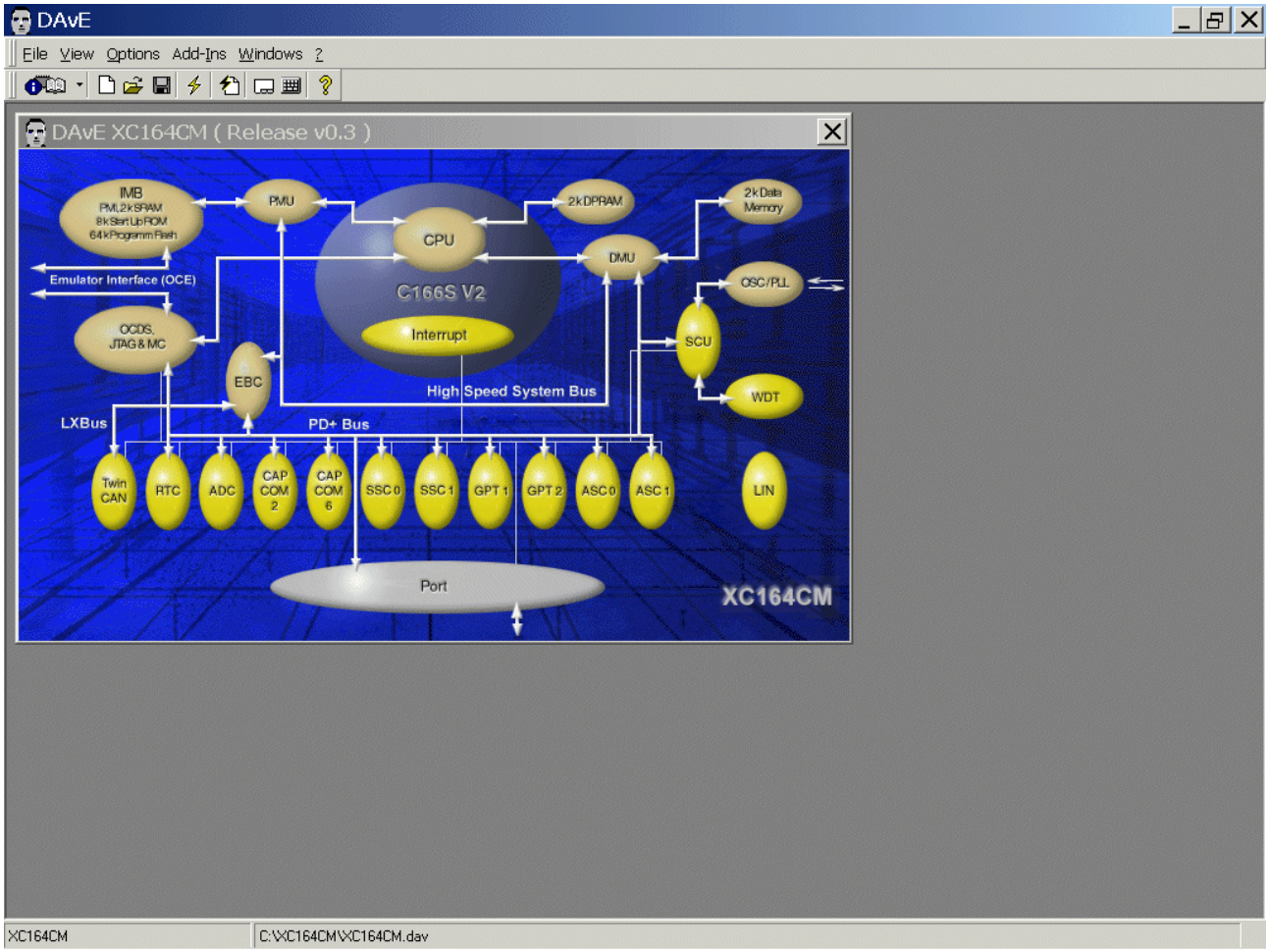


Start the program generator DAvE and open your XC164CM.dav DAvE project:

File
Open
Location: C:\XC164CM
Filename: XC164CM.dav

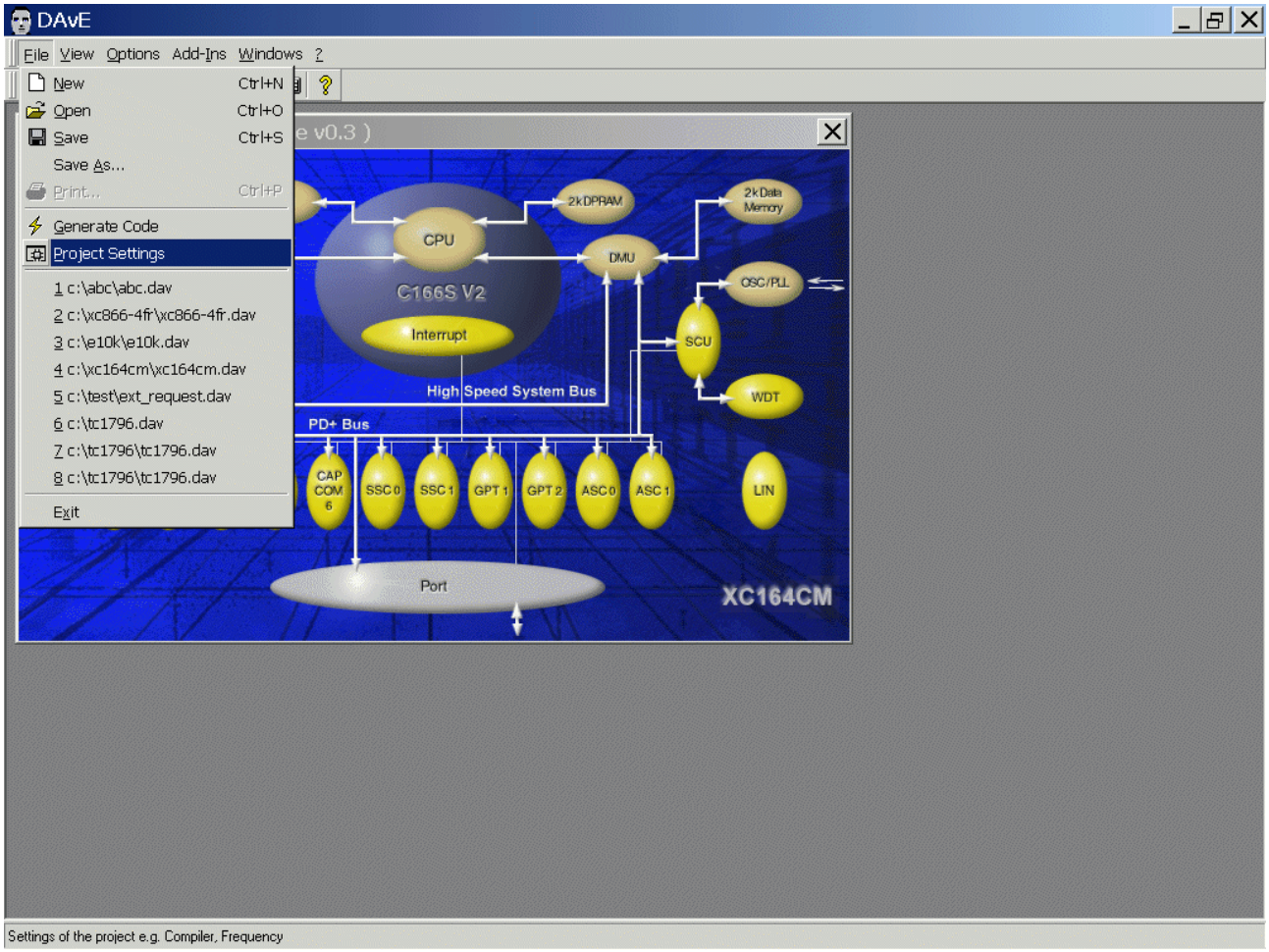


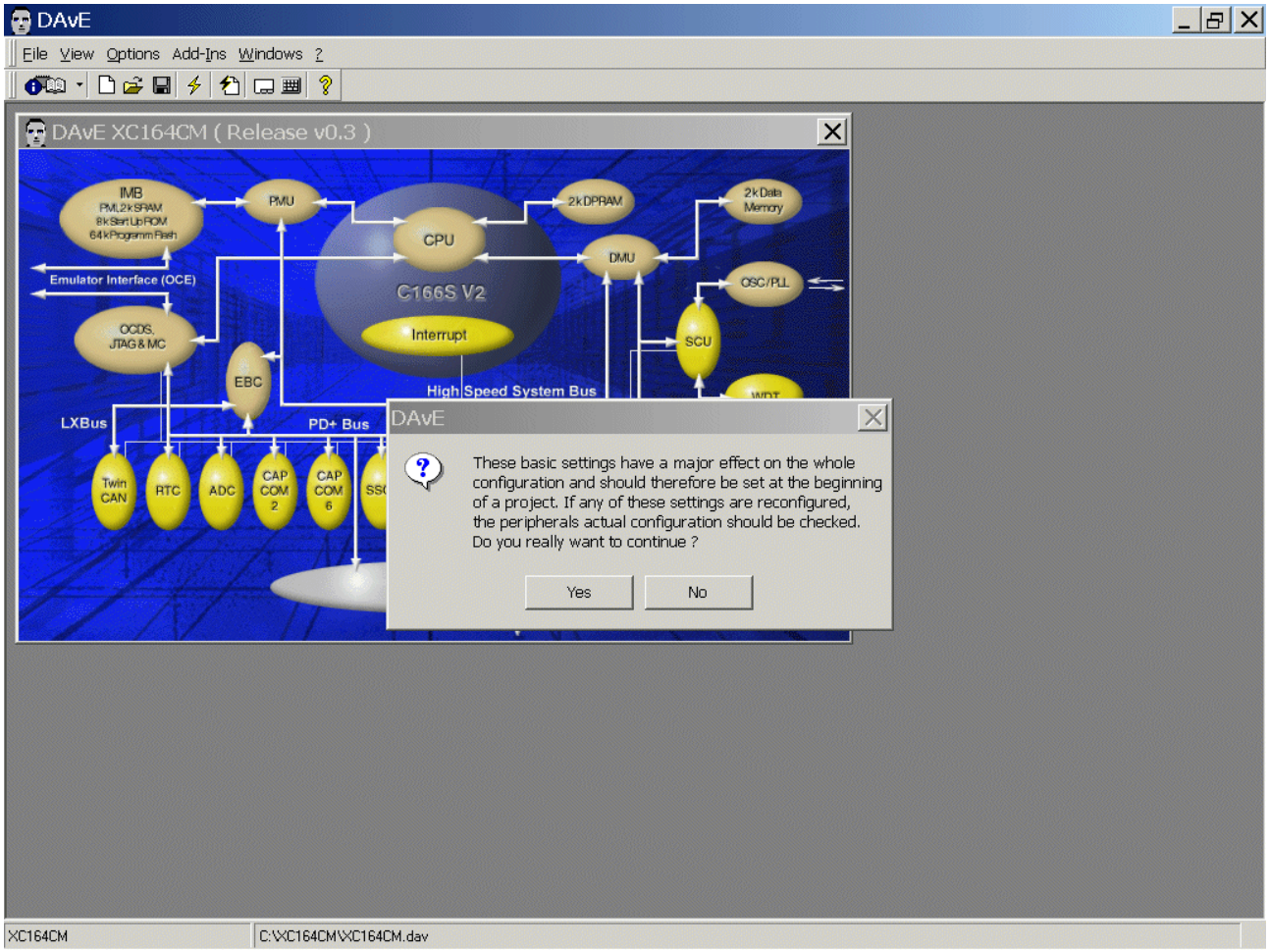
Click Open


DAvE
File View Options Add-Ins Windows ?
DAvE XC164CM (Release v0.3)
IMB
PMU
CPU
2k DPRAM
2k Data Memory
DMU
C166S V2
Interrupt
Emulator Interface (OCE)
OCDS, JTAG & MC
EBC
OSC/PLL
SCU
WDT
High Speed System Bus
LXBus
PD+ Bus
Twin CAN
RTC
ADC
CAP COM 2
CAP COM 6
SSC 0
SSC 1
GPT 1
GPT 2
ASC 0
ASC 1
LIN
Port
XC164CM
XC164CM
C:\XC164CM\XC164CM.dav

Change the Project Settings (configure as you can see in the screenshots):

File
Project Settings

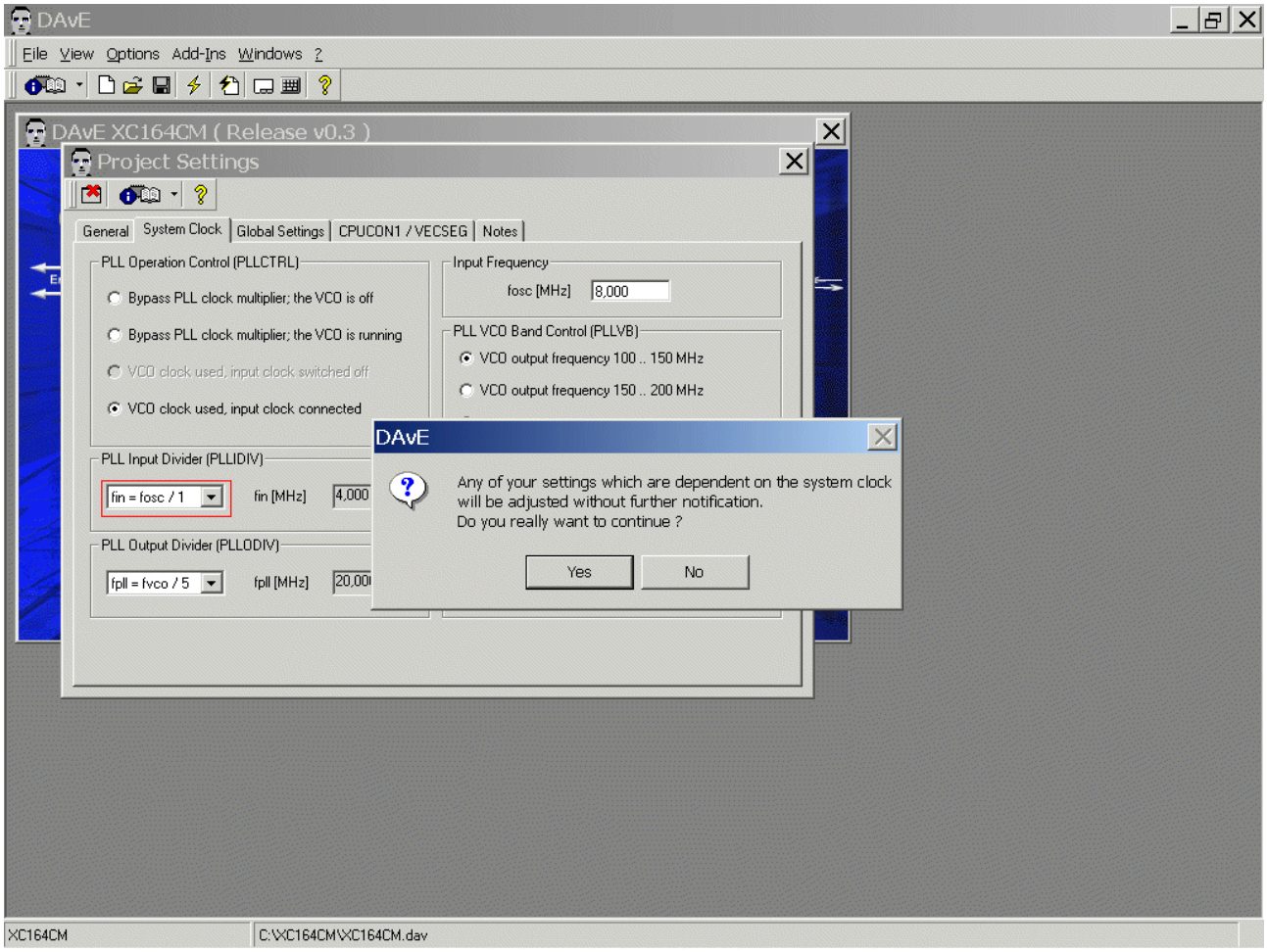



DAvE
File View Options Add-Ins Windows ?
DAvE XC164CM (Release v0.3)
IMB
PMU
CPU
2kDPRAM
2kData Memory
DMU
OSC/PLL
C166S V2
Interrupt
SCU
Emulator Interface (OCE)
OCDS, JTAG & MC
EBC
High Speed System Bus
LXBus
PD+ Bus
Twin CAN
RTC
ADC
CAP COM 2
CAP COM 6
DAvE
These basic settings have a major effect on the whole configuration and should therefore be set at the beginning of a project. If any of these settings are reconfigured, the peripherals actual configuration should be checked. Do you really want to continue ?
Yes
No
XC164CM
C:\XC164CM\XC164CM.dav

Click Yes

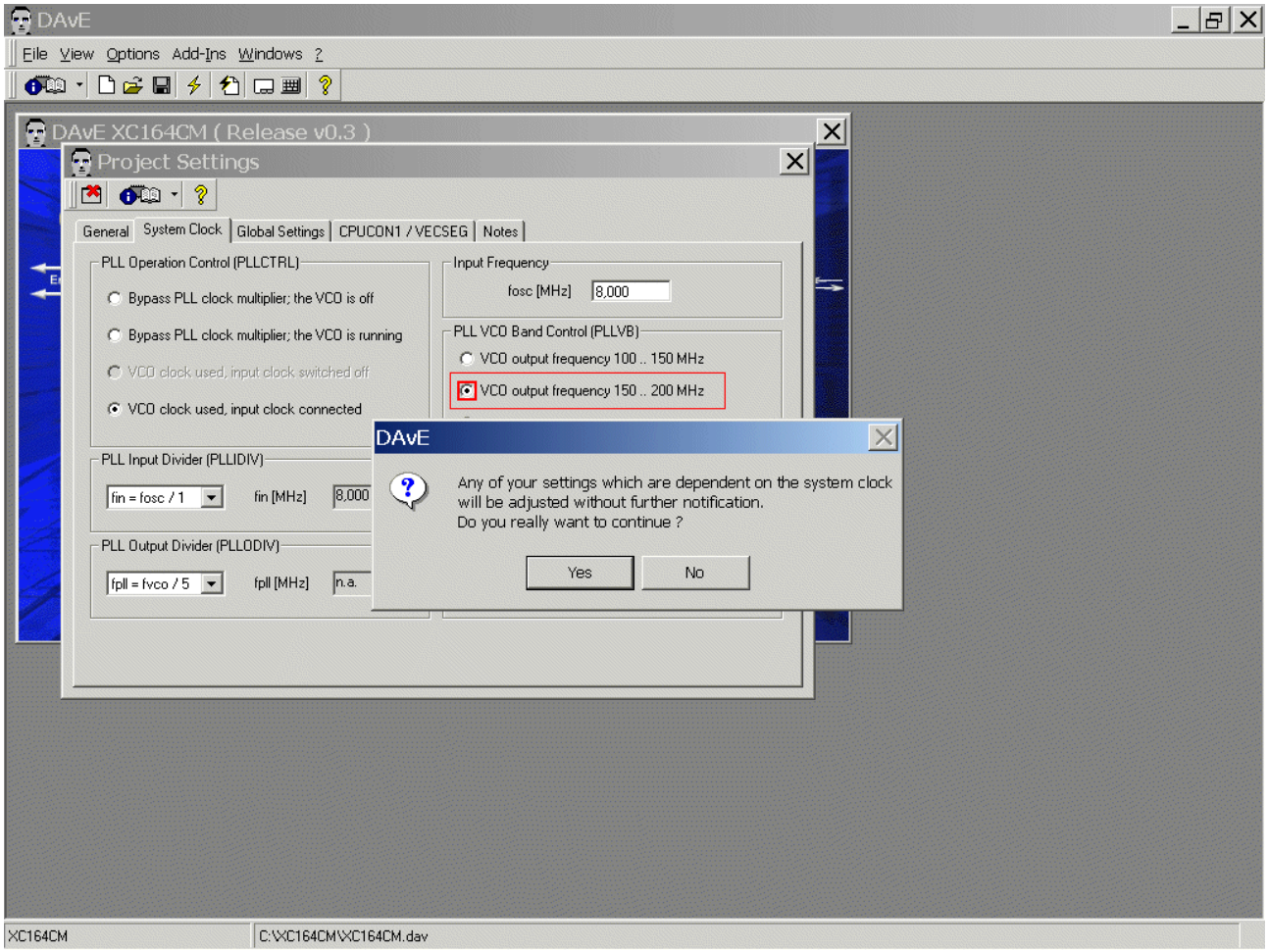
System Clock: CPU Clock will be 40 MHz :

System Clock: PLL Input Divider (PLLIDIV) select fin=fosc/1



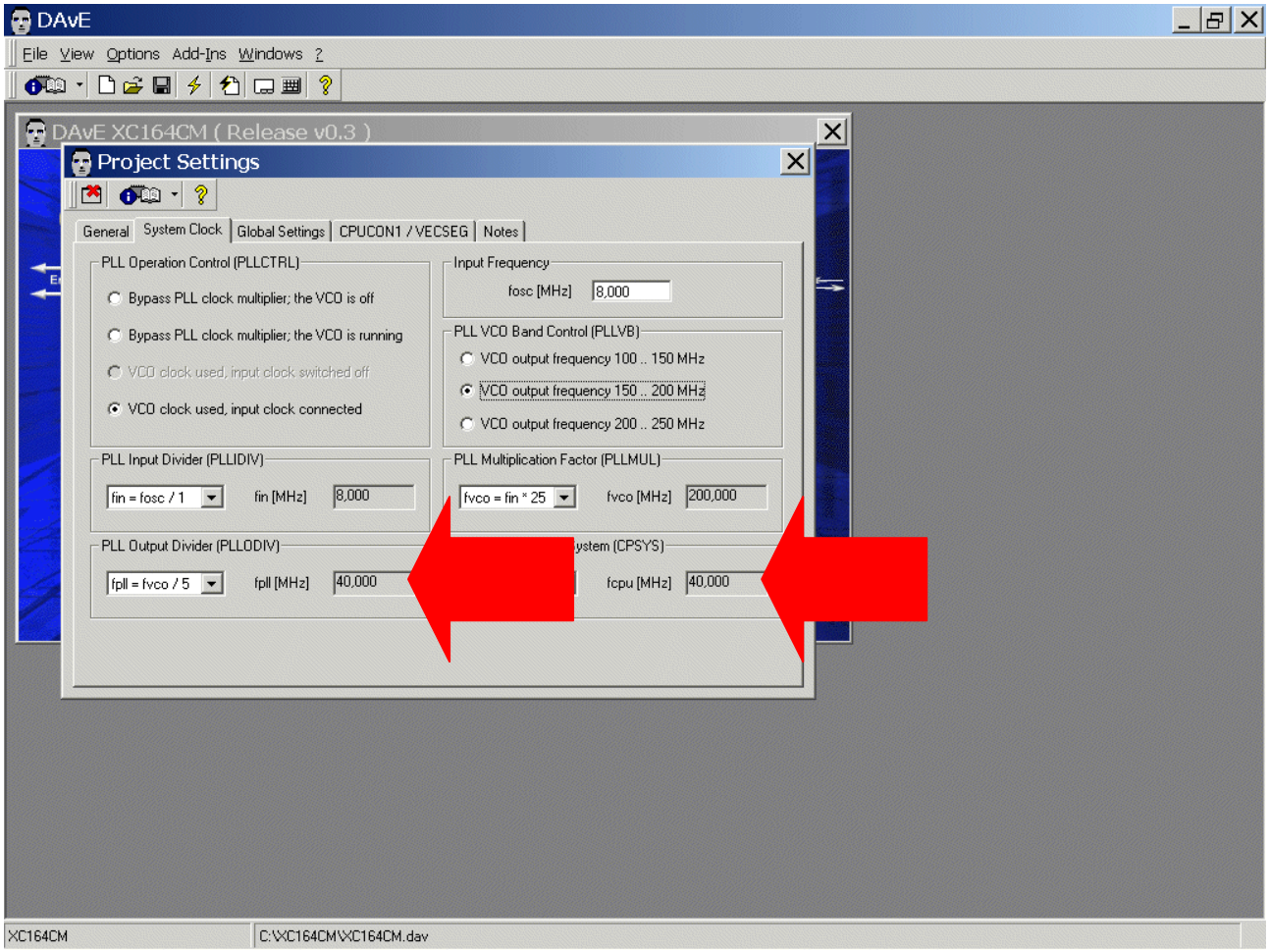
Click Yes

System Clock: PLL VCO Band Control (PLLVB) select ⊙ VCO output frequency 150..200 MHz

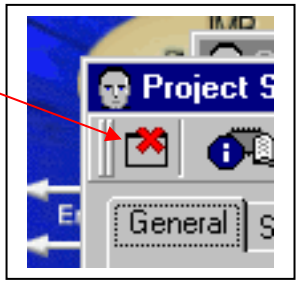


Click Yes

The CPU Clock is now 40 MHz:

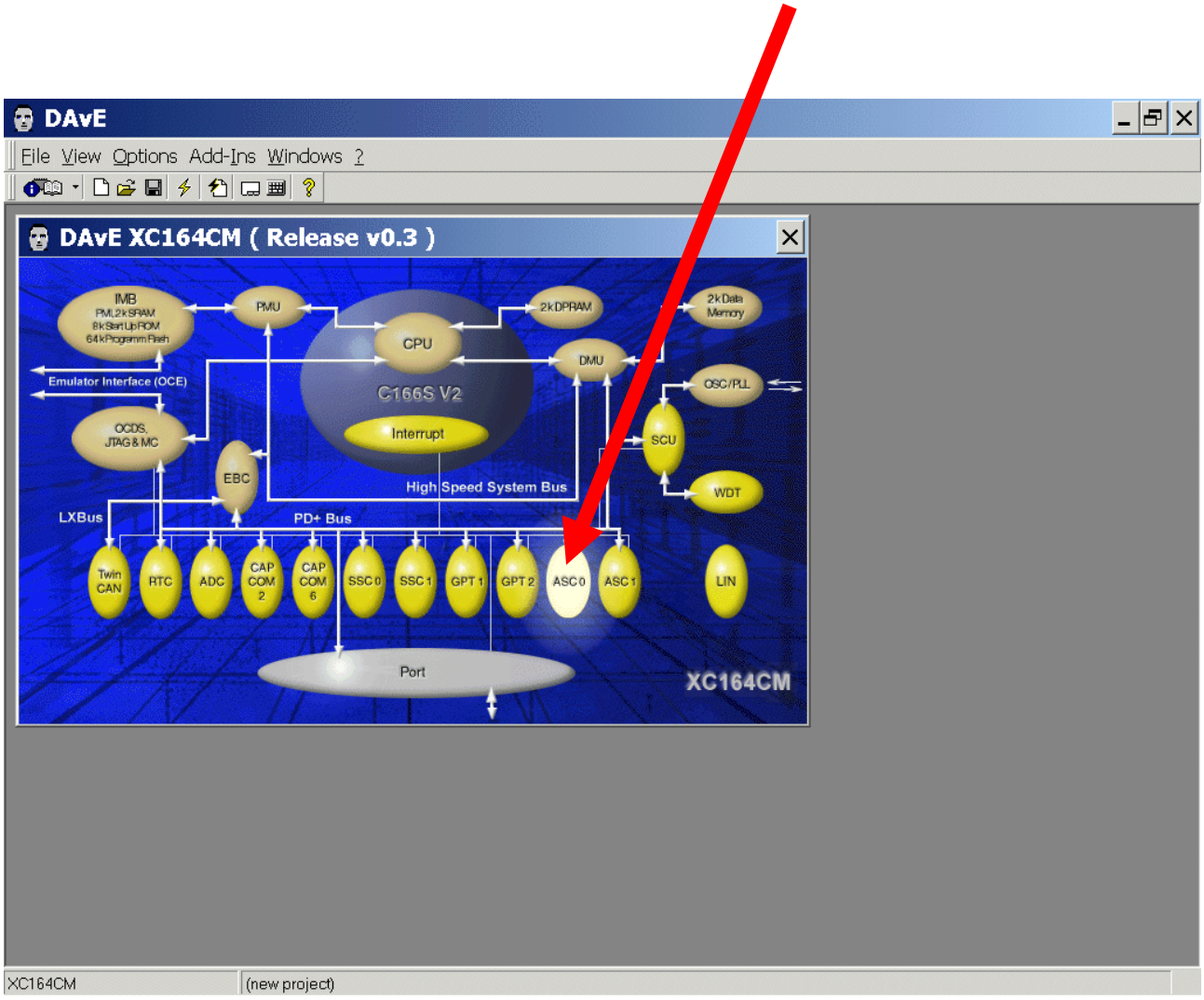


Exit this dialog now by clicking  the close button:

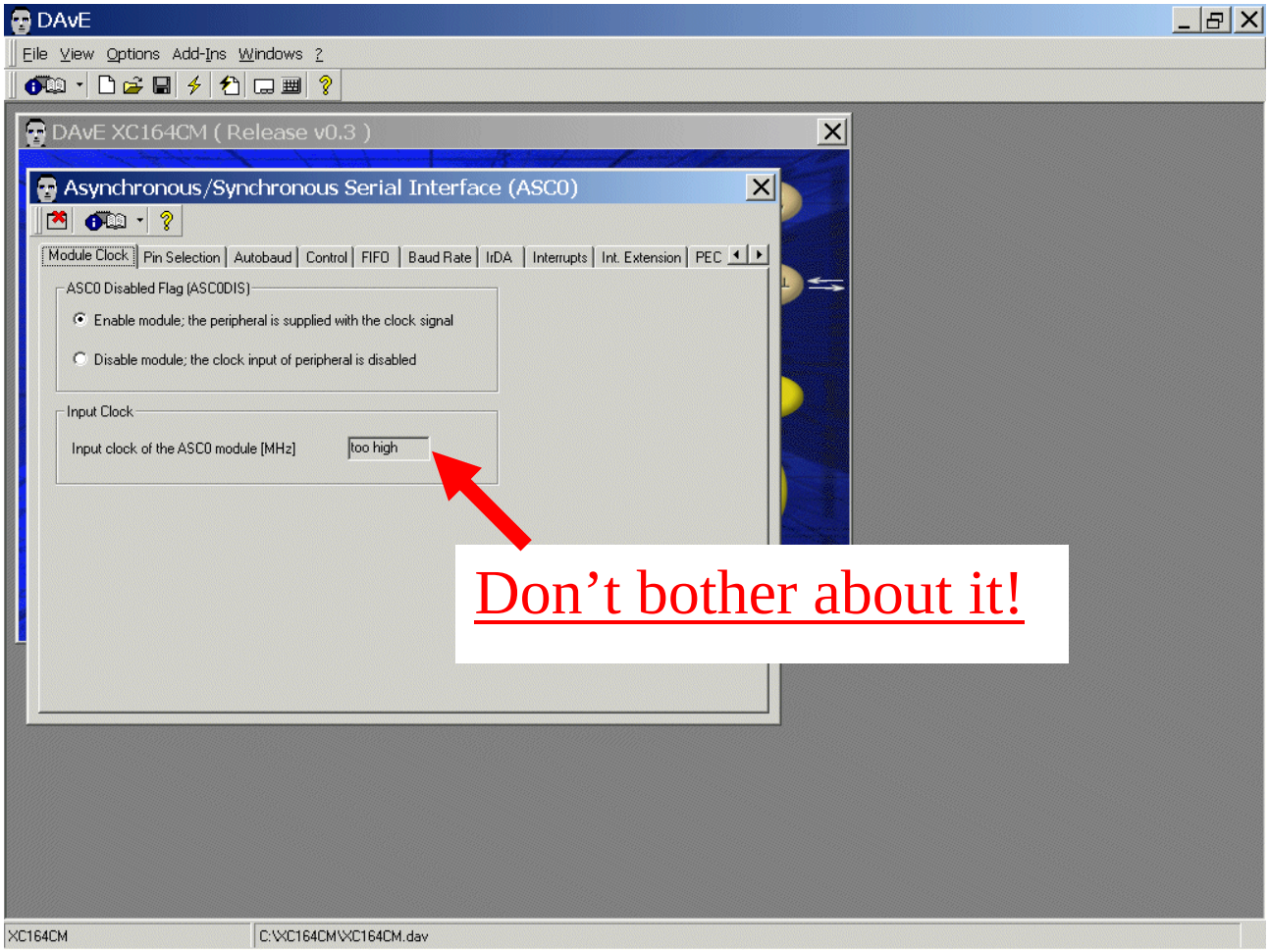


Reconfiguration of the ASC0:

The configuration window can be opened by clicking the specific block/module.

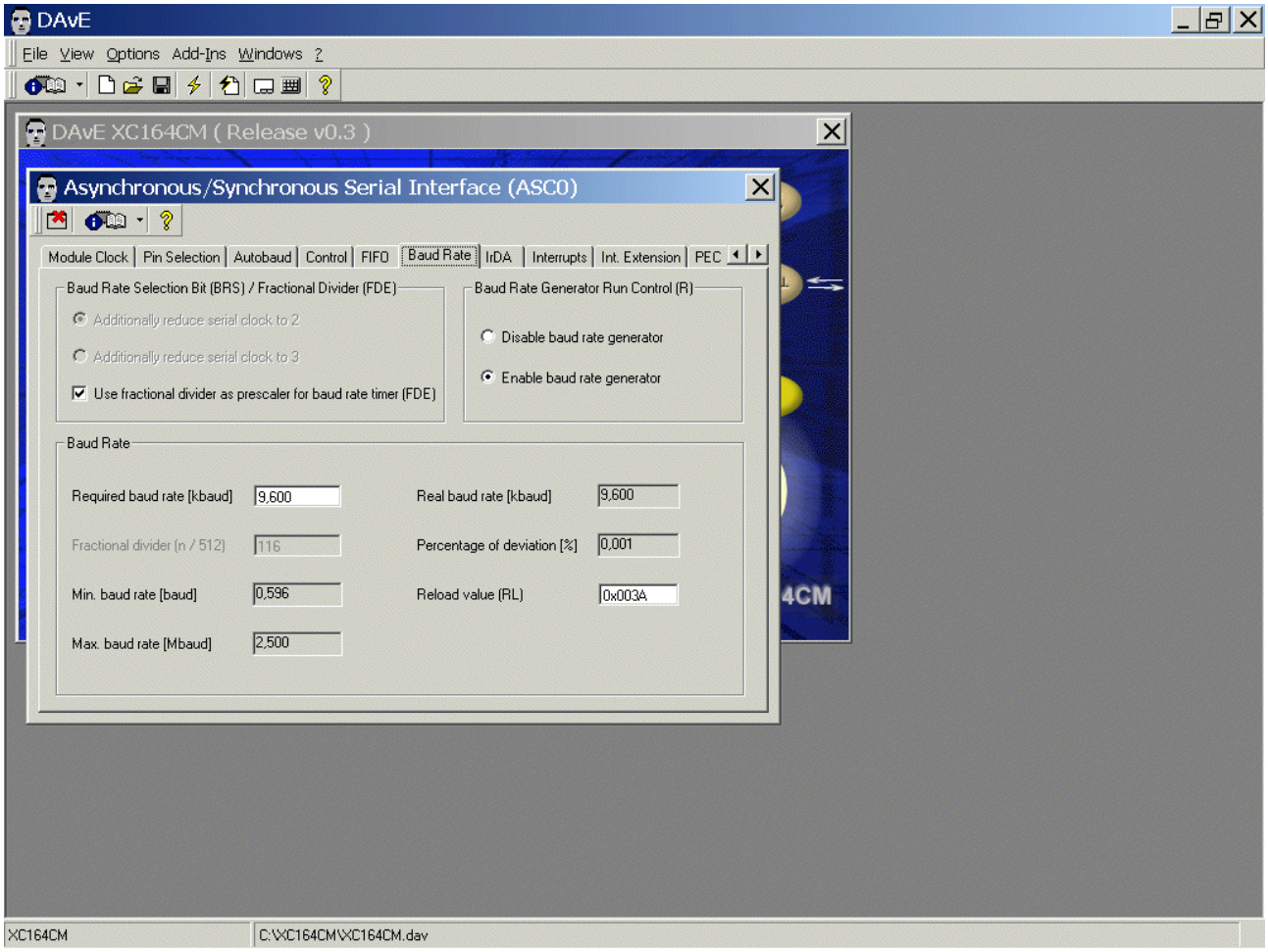


Module Clock: (do nothing)



Check / Compare:

Baud Rate: Baud Rate: Required baud rate [kbaud] input 9,600 <ENTER>
Baud Rate: Baud Rate Selection Bit: additionally – if you want: click ✓ Use fractional divider as prescaler ...

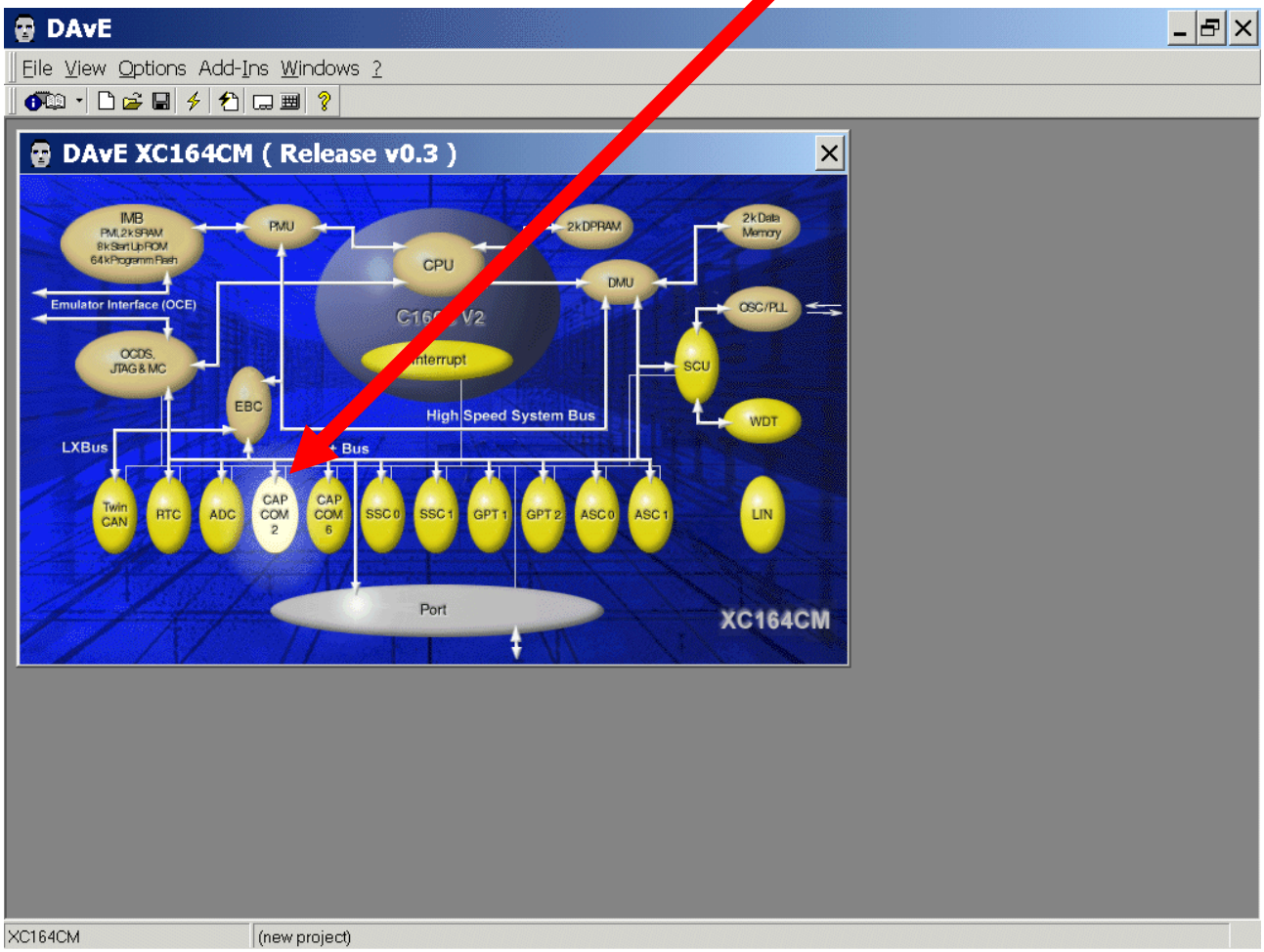


Note:
Validate each alpha numeric entry by pressing ENTER.

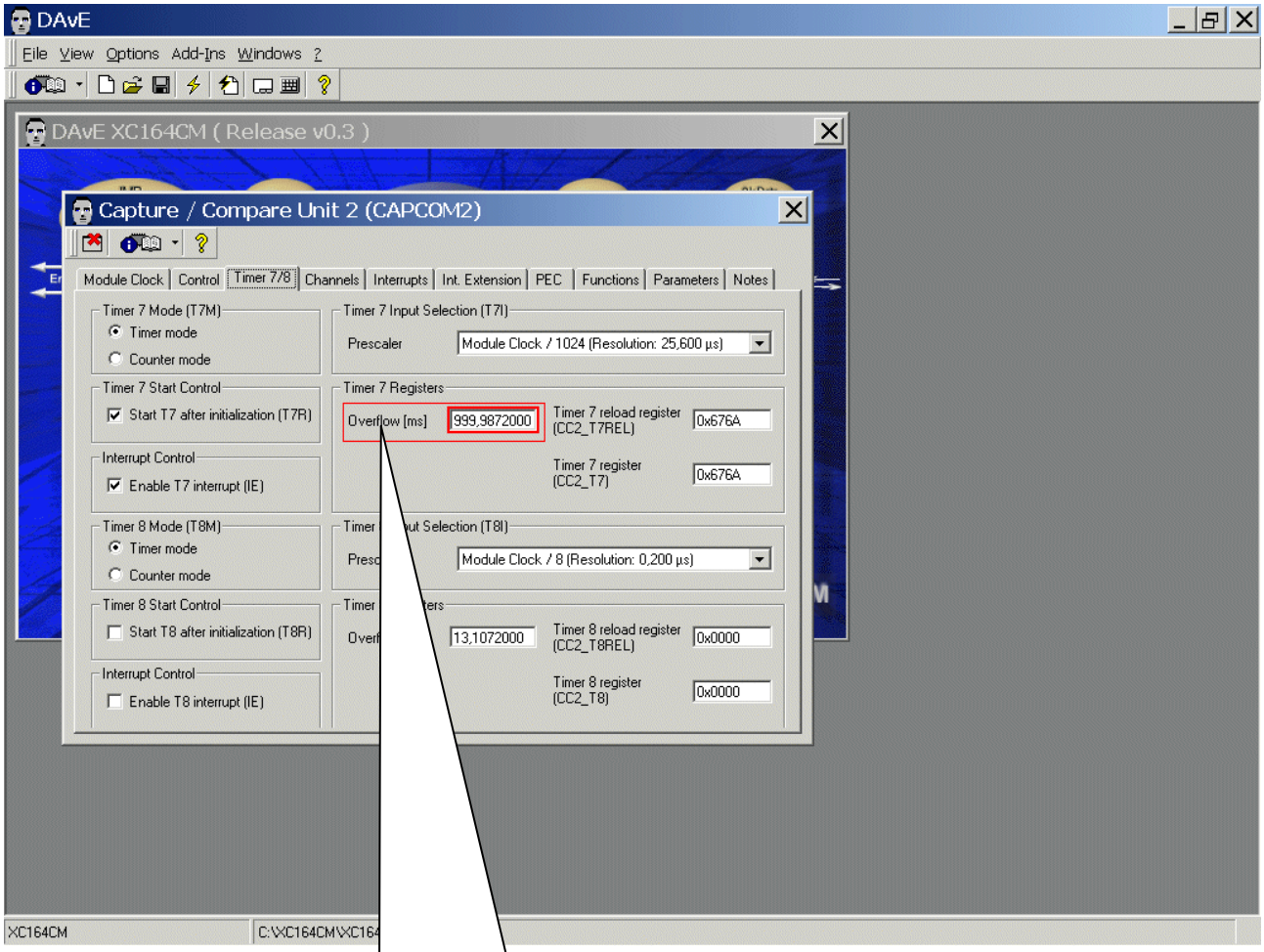
Exit this dialog now by clicking the close button.

Reconfigure Timer T7 in CAPCOM 2:

The configuration window can be opened by clicking the specific block/module.



Timer 7/8: Timer 7 Registers: Overflow [ms]: insert 1000 <ENTER> (for 1 second)

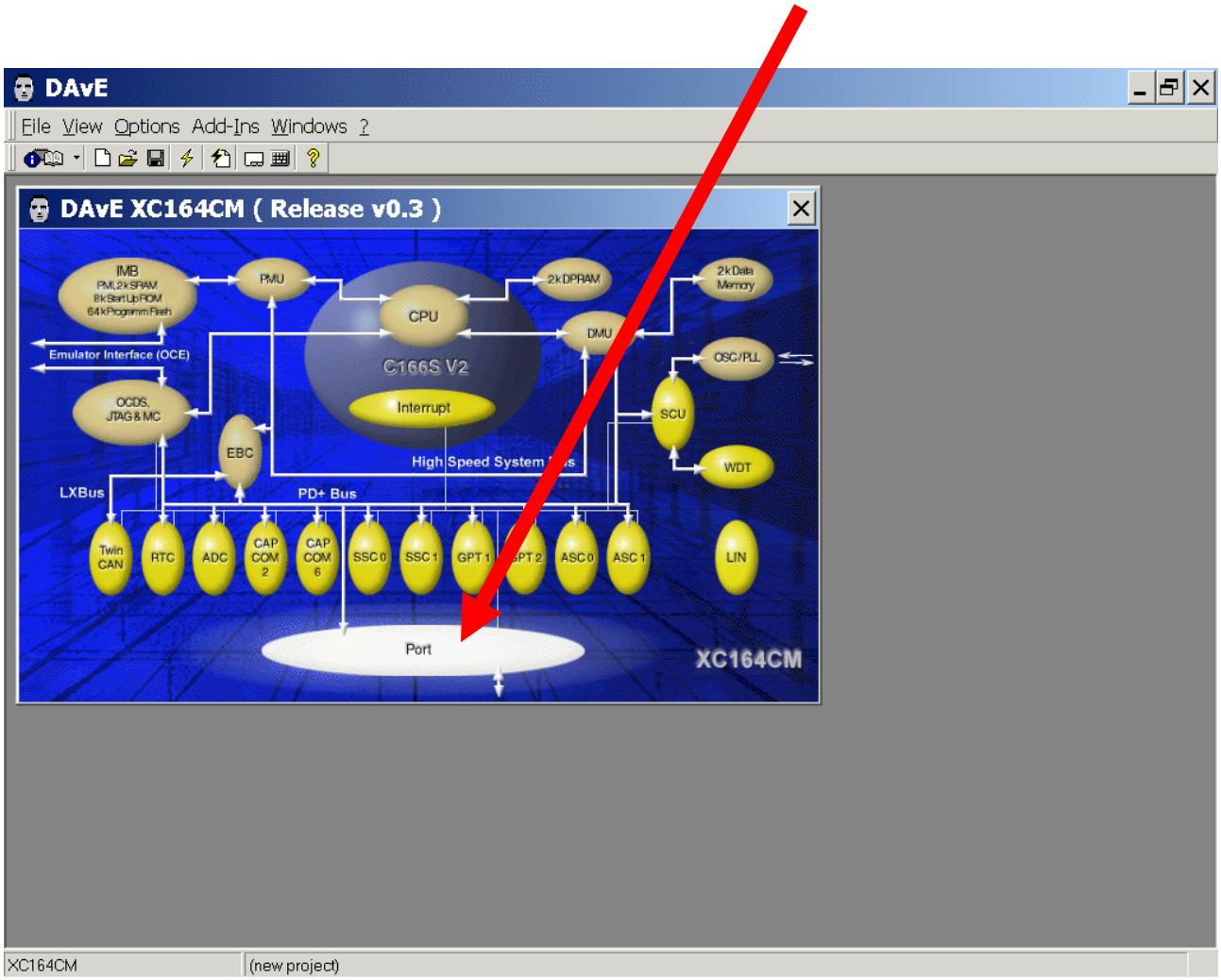


DAvE calculates automatically the nearest possible solution (999,98 ms) for the inserted value (1000 ms) - accordingly to the selected Timer 7 configuration.

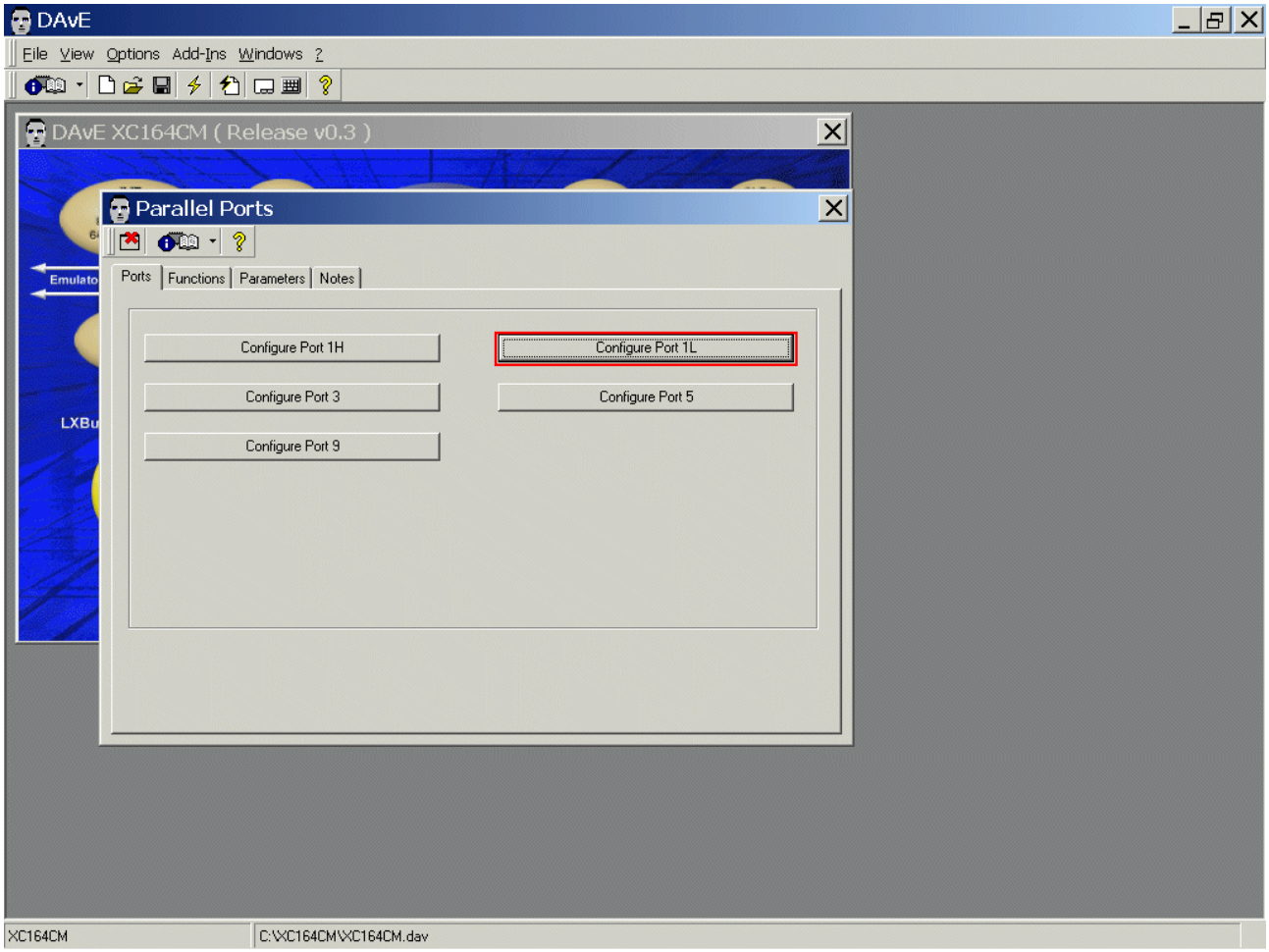
Exit this dialog now by clicking  the close button.

Reconfigure Port 1L:

The configuration window can be opened by clicking the specific block/module.



Ports: click Configure Port 1L

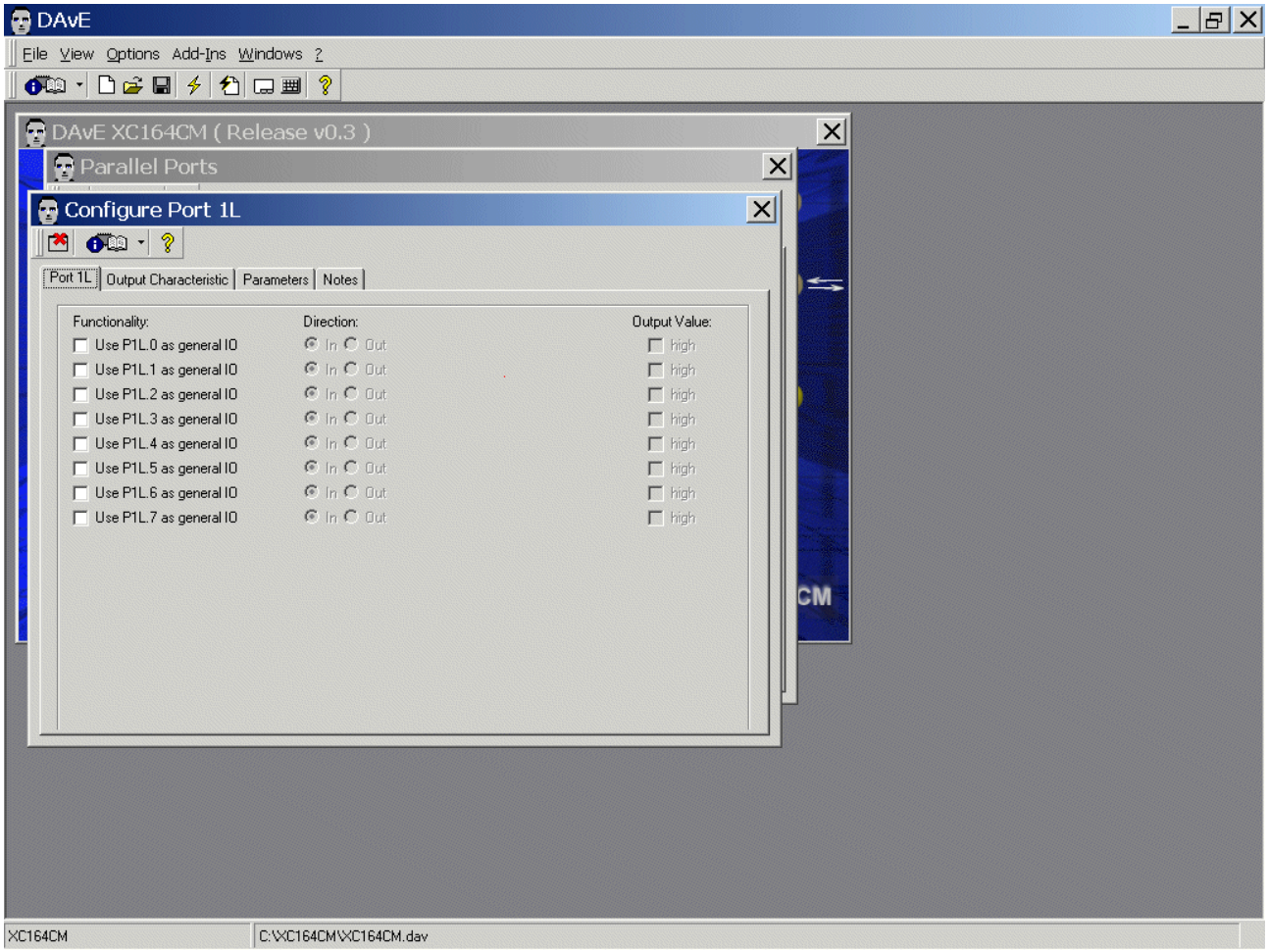


Ports: click ☐ Configure Port 1L
Port 1L: Functionality: click to unselect ☐ Use P1L.1 as general IO

Ports: click ☐ Configure Port 1L
Port 1L: Functionality: click to unselect ☐ Use P1L.3 as general IO

Ports: click ☐ Configure Port 1L
Port 1L: Functionality: click to unselect ☐ Use P1L.5 as general IO

Ports: click ☐ Configure Port 1L
Port 1L: Functionality: click to unselect ☐ Use P1L.7 as general IO

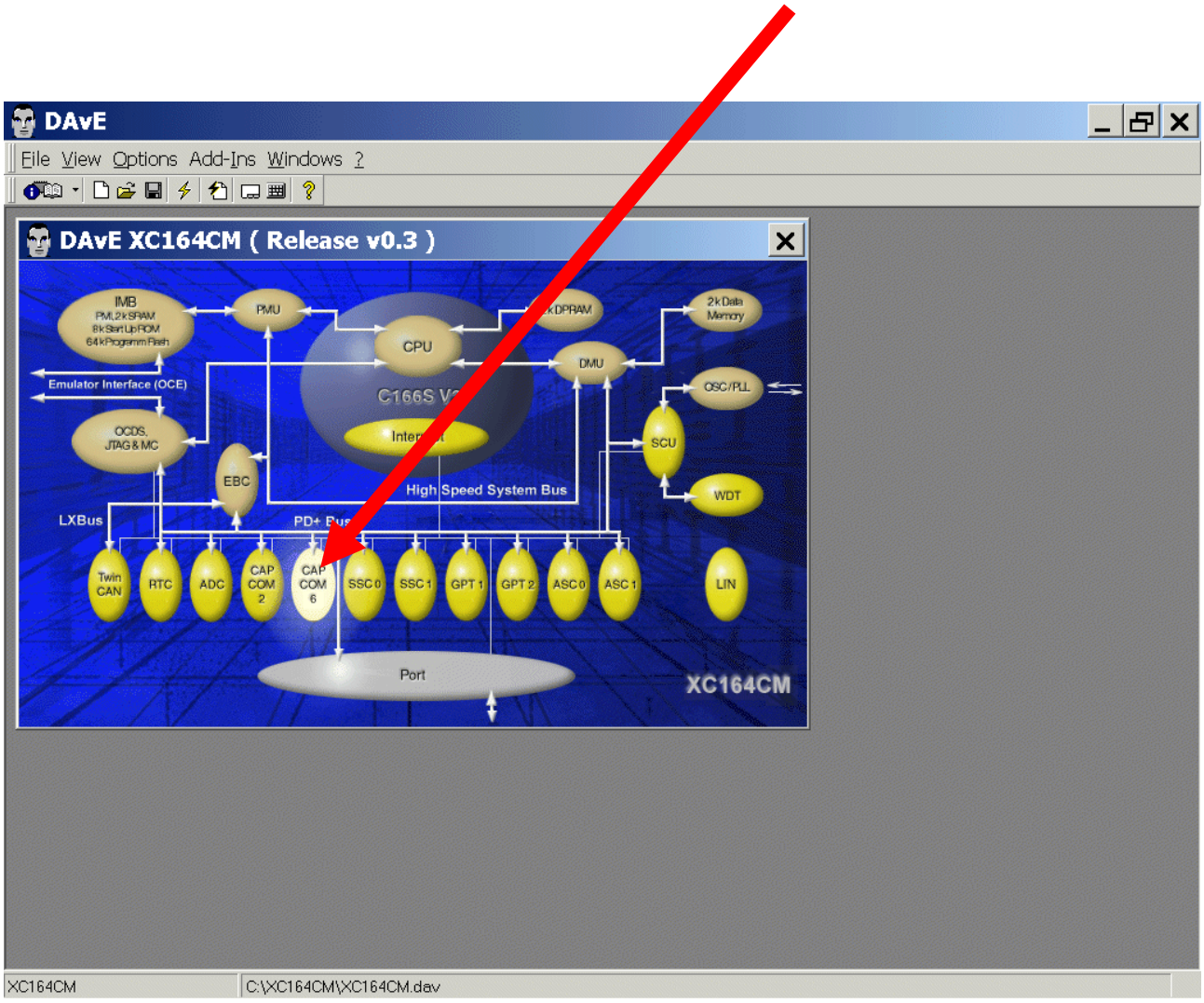


Exit this dialog now by clicking the close button.

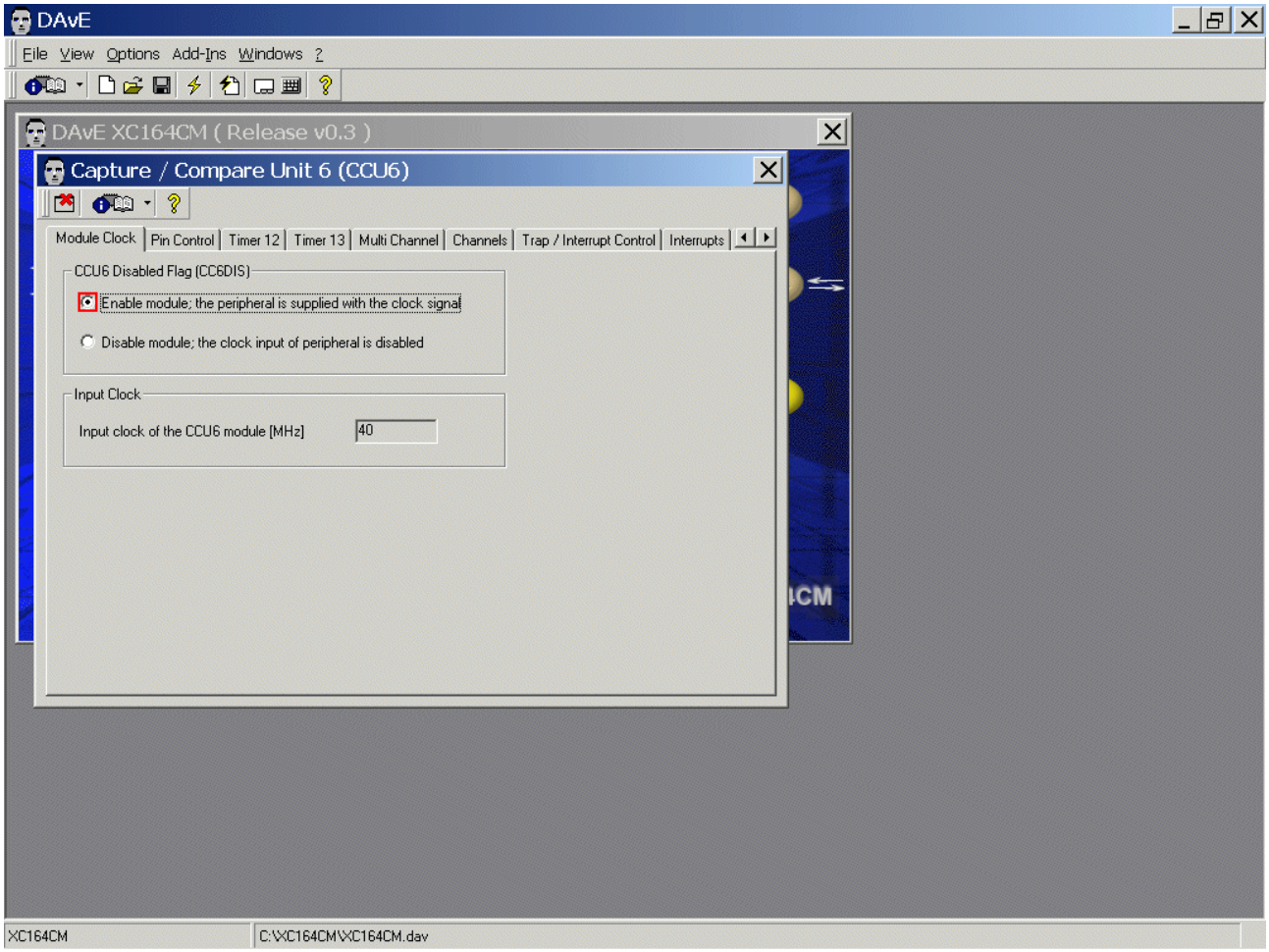
Exit this dialog now by clicking the close button.

Configuration of the CAPCOM6 (configure as you can see in the screenshots):

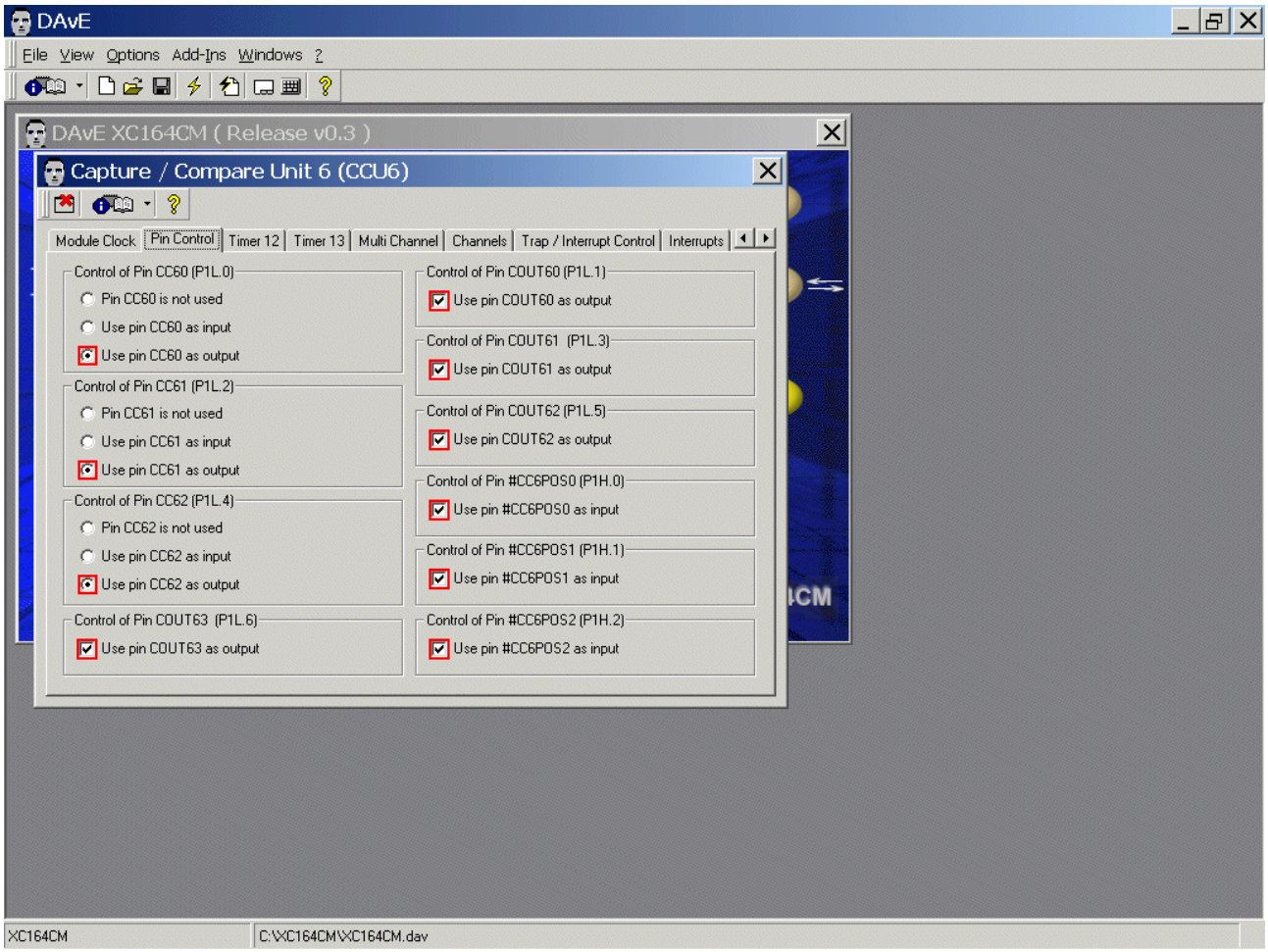
The configuration window can be opened by clicking the specific block/module.



CCU6: Module Clock: CCU6 Disabled Flag: click ⊙ Enable module

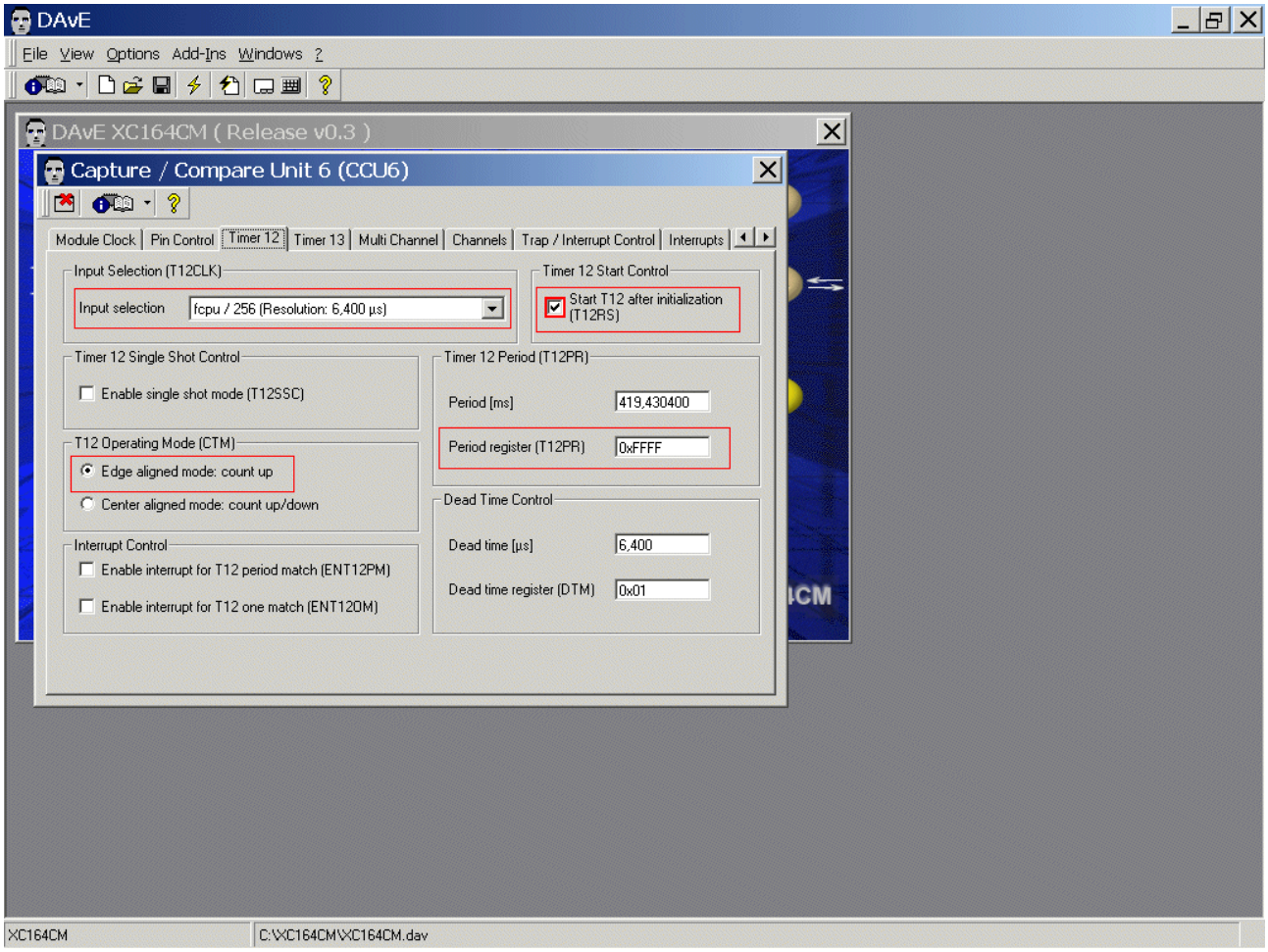


CCU6: Pin Control: Control of Pin CC60: click ⦿ Use pin CC60 as output
CCU6: Pin Control: Control of Pin CC61: click ⦿ Use pin CC61 as output
CCU6: Pin Control: Control of Pin CC62: click ⦿ Use pin CC62 as output
CCU6: Pin Control: Control of Pin COUT63: click ✓ Use pin COUT63 as output
CCU6: Pin Control: Control of Pin COUT60: click ✓ Use pin COUT60 as output
CCU6: Pin Control: Control of Pin COUT 61: click ✓ Use pin COUT61 as output
CCU6: Pin Control: Control of Pin COUT 62: click ✓ Use pin COUT62 as output
CCU6: Pin Control: Control of Pin #CC6POS0: click ✓ Use pin CC6POS0 as input
CCU6: Pin Control: Control of Pin #CC6POS1: click ✓ Use pin CC6POS1 as input
CCU6: Pin Control: Control of Pin #CC6POS2: click ✓ Use pin CC6POS2 as input



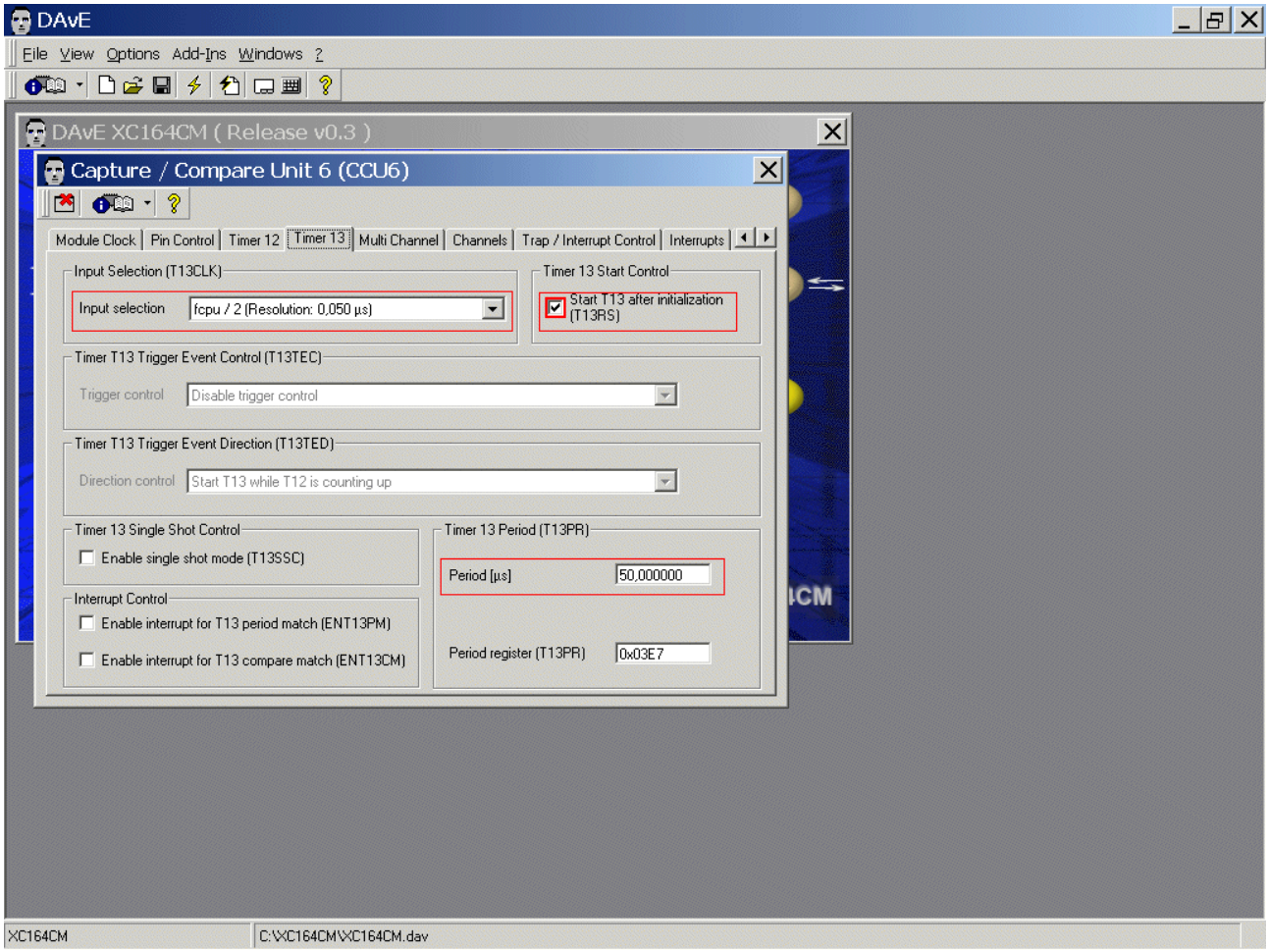
Timer T12 Configuration:

CCU6: T12: Input selection (T12CLK): Input selection: select fcpu/256 (Resolution: 6,400 µs)
CCU6: T12: Timer Operation Mode: click/check ⊙ Edge aligned mode: count up
CCU6: T12: Timer 12 Start Control: click ✓ Start T12 after initialization
CCU6: T12: Timer 12 Period: Period register (T12PR) insert 0xFFFF <ENTER>

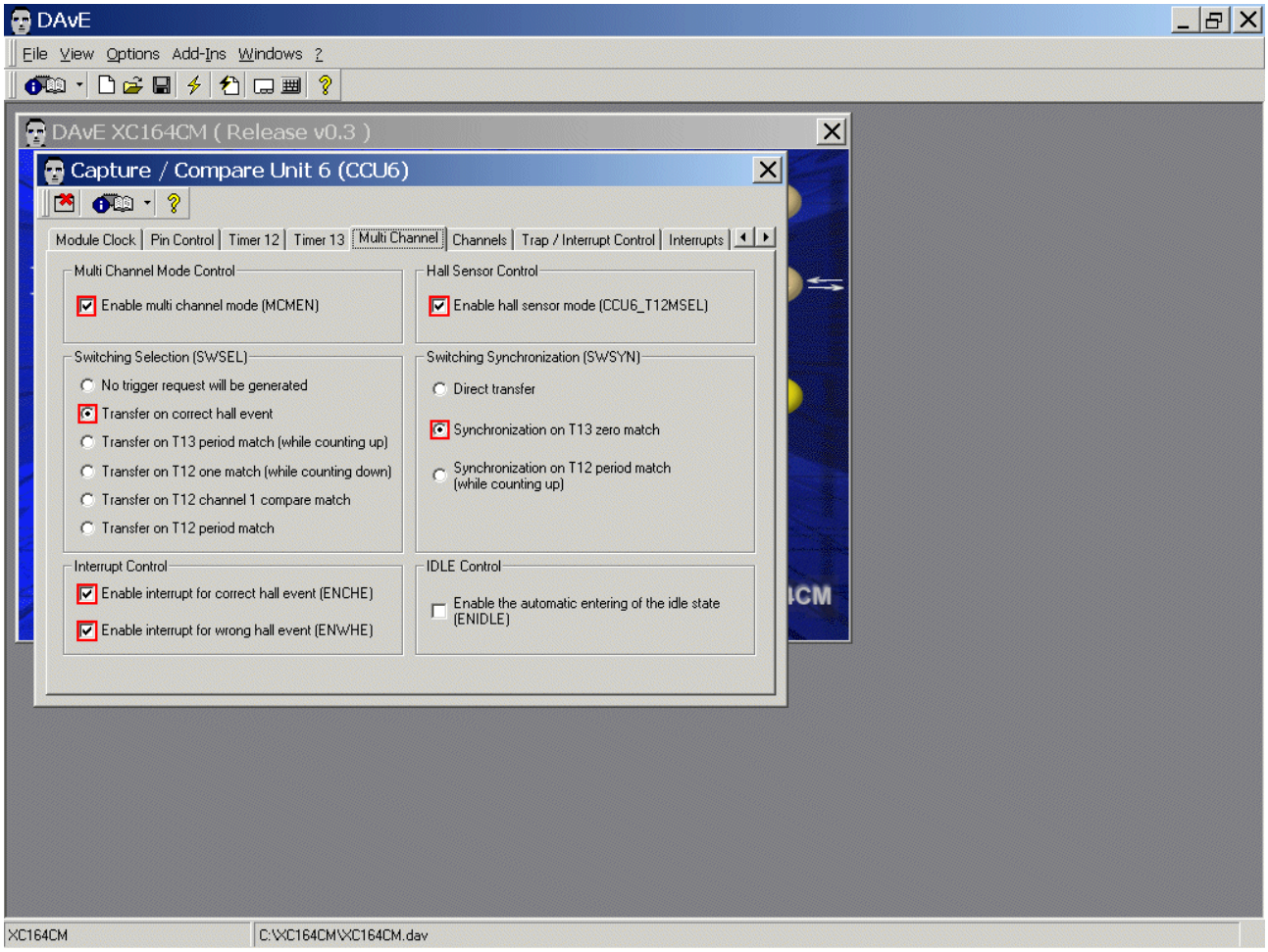


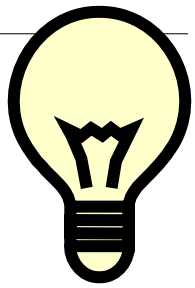
Timer T13 Configuration:

CCU6: T13: Input Selection: Input selection select fcpu/2 (Resolution: 0,050 µs)
CCU6: T13: Timer 13 Start Control: click ✓ Start T13 after initialization (T13RS)
CCU6: T13: Timer 13 Period: insert 50000 <ENTER> (Period [µs] = 50,000000)



CCU6: Multi Channel: Multi Channel Mode Control: click ✓ Enable multi channel mode
CCU6: Multi Channel: Switching Selection: click ⊙ Transfer on correct hall event
CCU6: Multi Channel: Interrupt Control: click ✓ Enable interrupt for correct hall event
CCU6: Multi Channel: Interrupt Control: click ✓ Enable interrupt for wrong hall event
CCU6: Multi Channel: Hall Sensor Control: click ✓ Enable hall sensor mode
CCU6: Multi Channel: Switching Synchronization: click ⊙ Synchronisation on T13 zero match

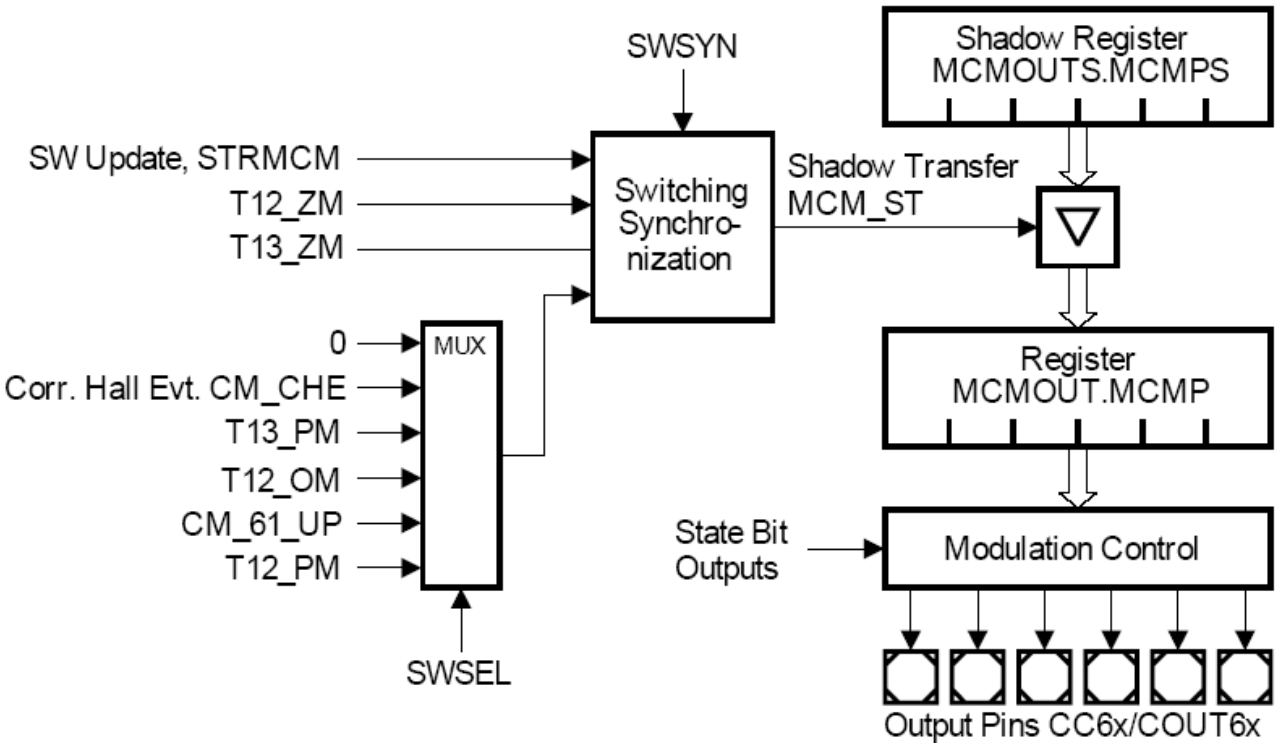


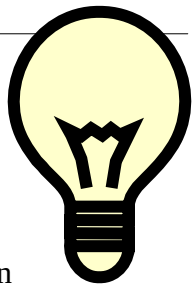


Register Information / Additional Information:

Multi-Channel Mode:

The Multi-Channel mode offers the possibility to modulate all six T12-related output signals with one instruction.
The bits in bitfield MCMOUT.MCMP are used to specify the outputs which may become active.

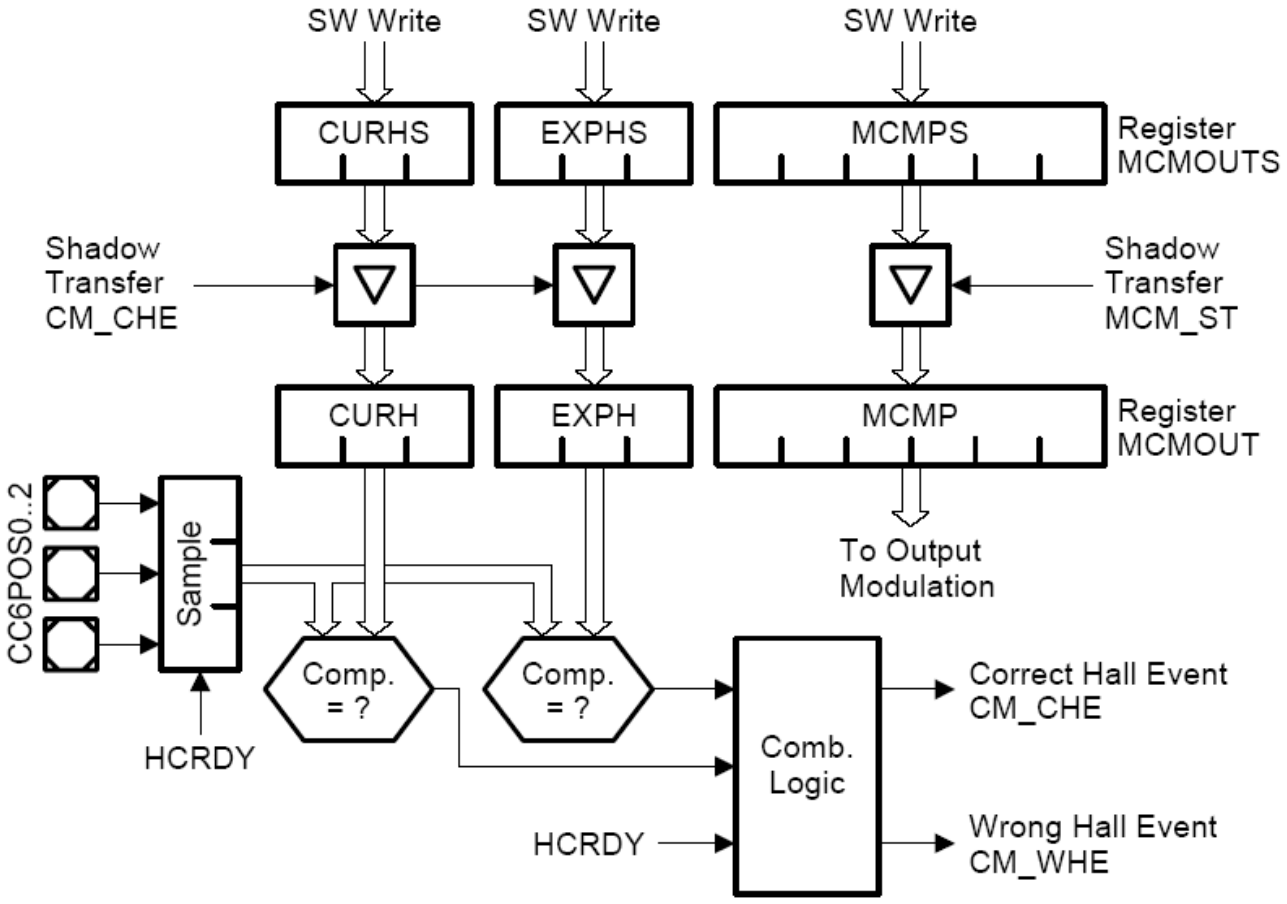




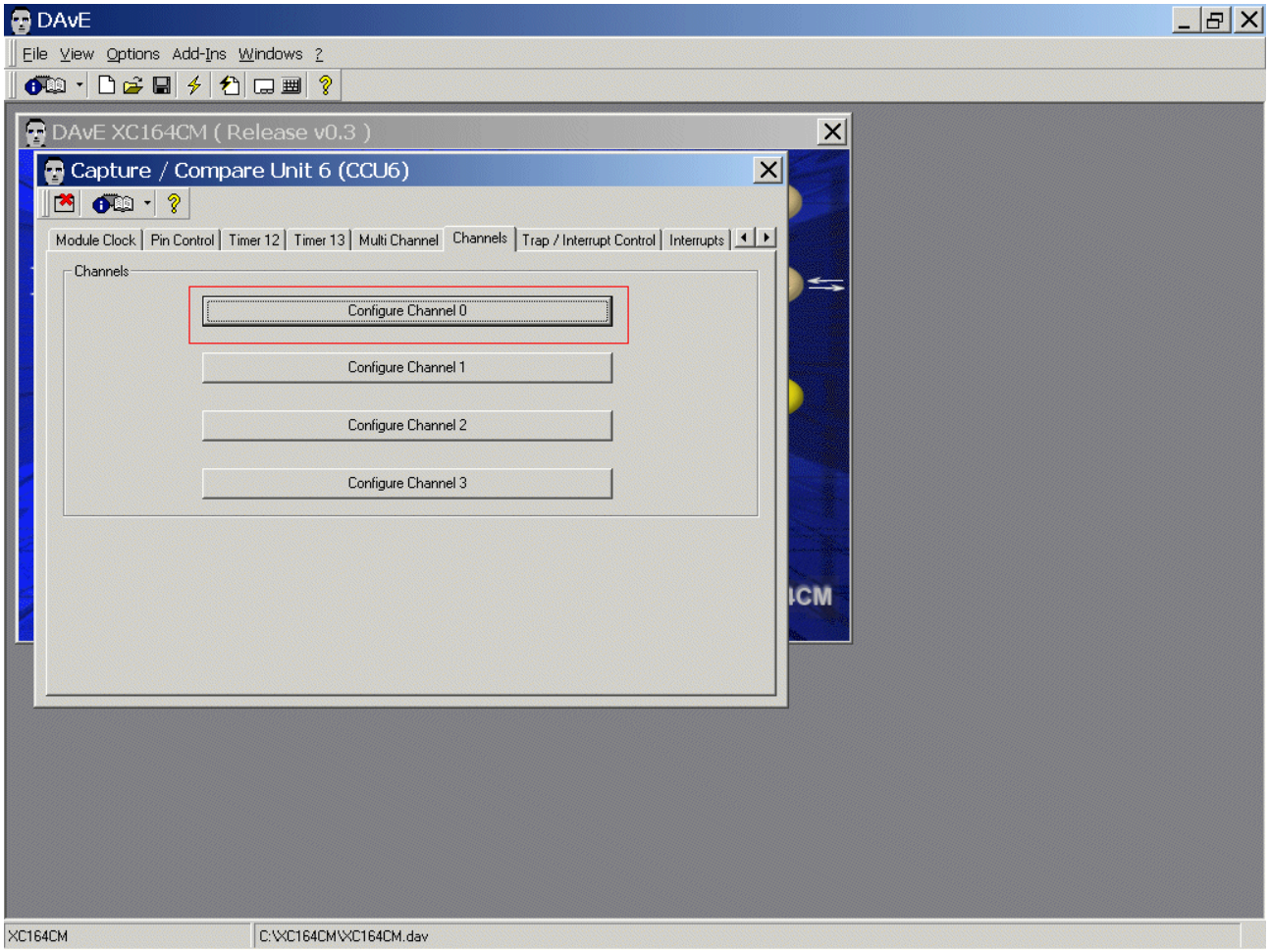
Register Information / Additional Information:

Hall Sensor Mode:

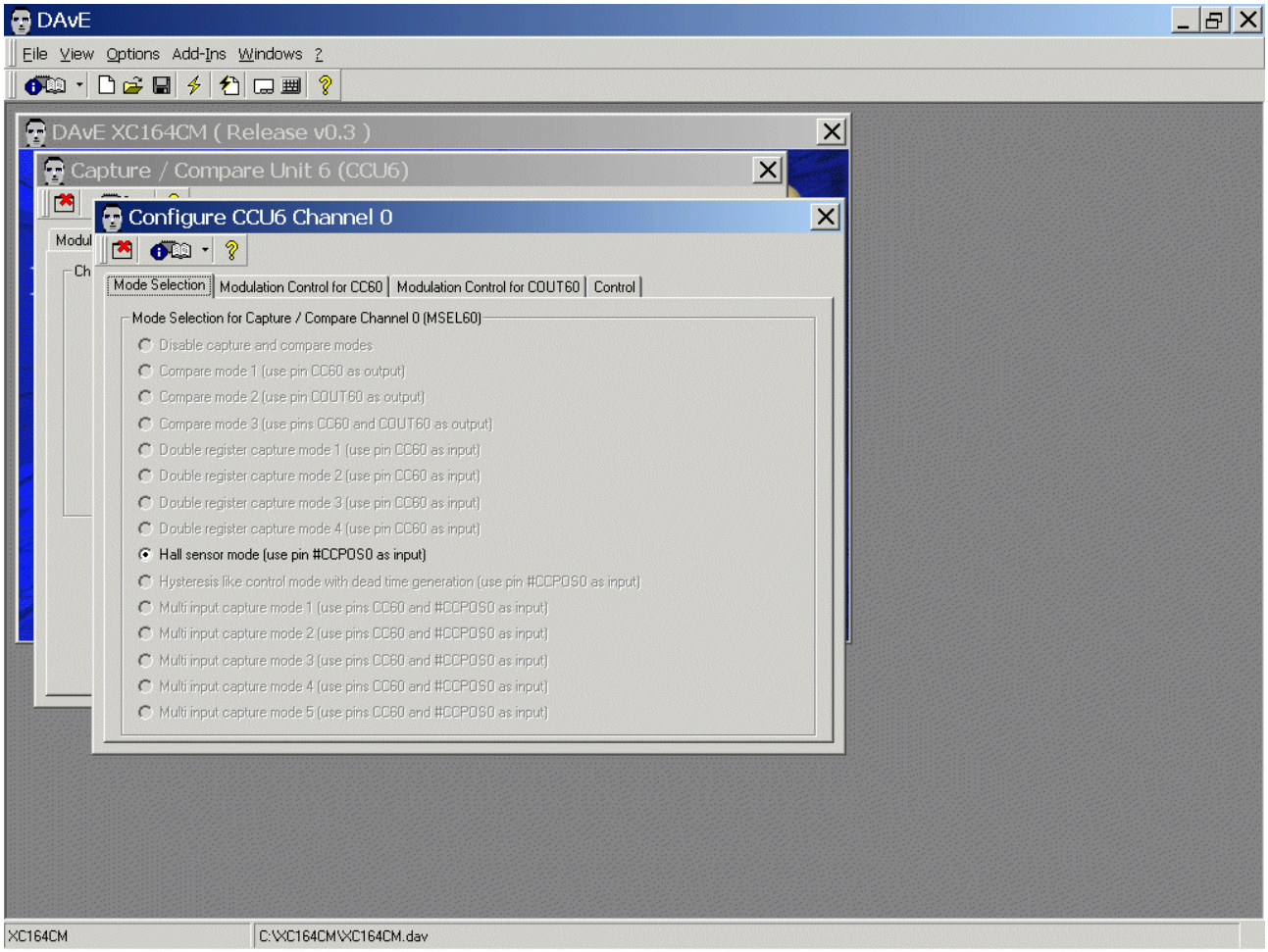
For Brushless DC-Motors, usually the Multi-Channel Mode is used, as the modulation patterns need to be output to properly control the motor. These patterns need to be output in relation to the angular position of the motor. For this, usually Hall sensors or Back-EMF sensing are used to determine the angular rotor position. The CAPCOM6 provides three inputs, CC6POS0 … CC6POS2, which can be used as inputs for the Hall sensors or the Back-EMF detection signals.
There is a strong correlation between the motor position and the output modulation pattern. When a certain position of the motor has been reached, indicated by the sampled Hall sensor inputs (the Hall pattern), the next, pre-determined modulation pattern has to be output. Because of different machine types, the modulation pattern for driving the motor can vary. Therefore, it is wishful to have a wide flexibility in defining the correlation between the Hall pattern and the corresponding modulation pattern.
The CAPCOM6 offers this by having a register which contains the actual current Hall pattern (CURH), the next expected Hall pattern (EXPH) and the corresponding output pattern (MCMP). A new modulation pattern is output when the sampled Hall inputs match the expected ones (EXPH). To detect the next rotation phase (segment for block commutation), the CAPCOM6 monitors the Hall inputs for changes. When the next expected Hall pattern is detected, the next corresponding modulation pattern is output.



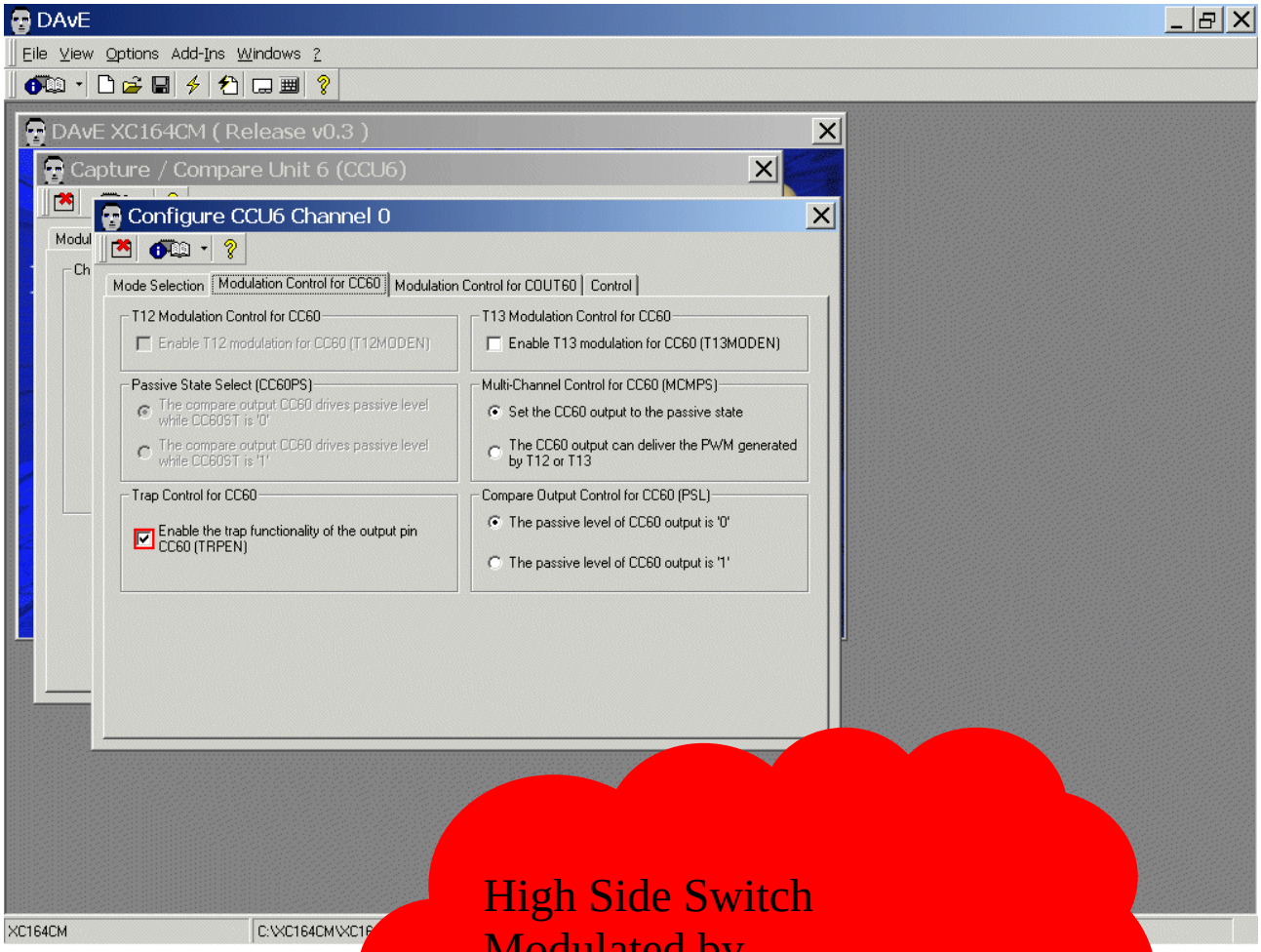
CCU6: Channels: click Configure Channel 0:



CCU6: Channels: Configure Channel 0:
Mode Selection: Mode Selection for Capture / Compare Channel 0: check Hall sensor mode



CCU6: Channels: Configure Channel 0: Modulation Control for CC60:
Trap Control for CC60: click ✓ Enable the trap functionality of the output pin CC60
(Enable the TRAP functionality for emergency shut down)

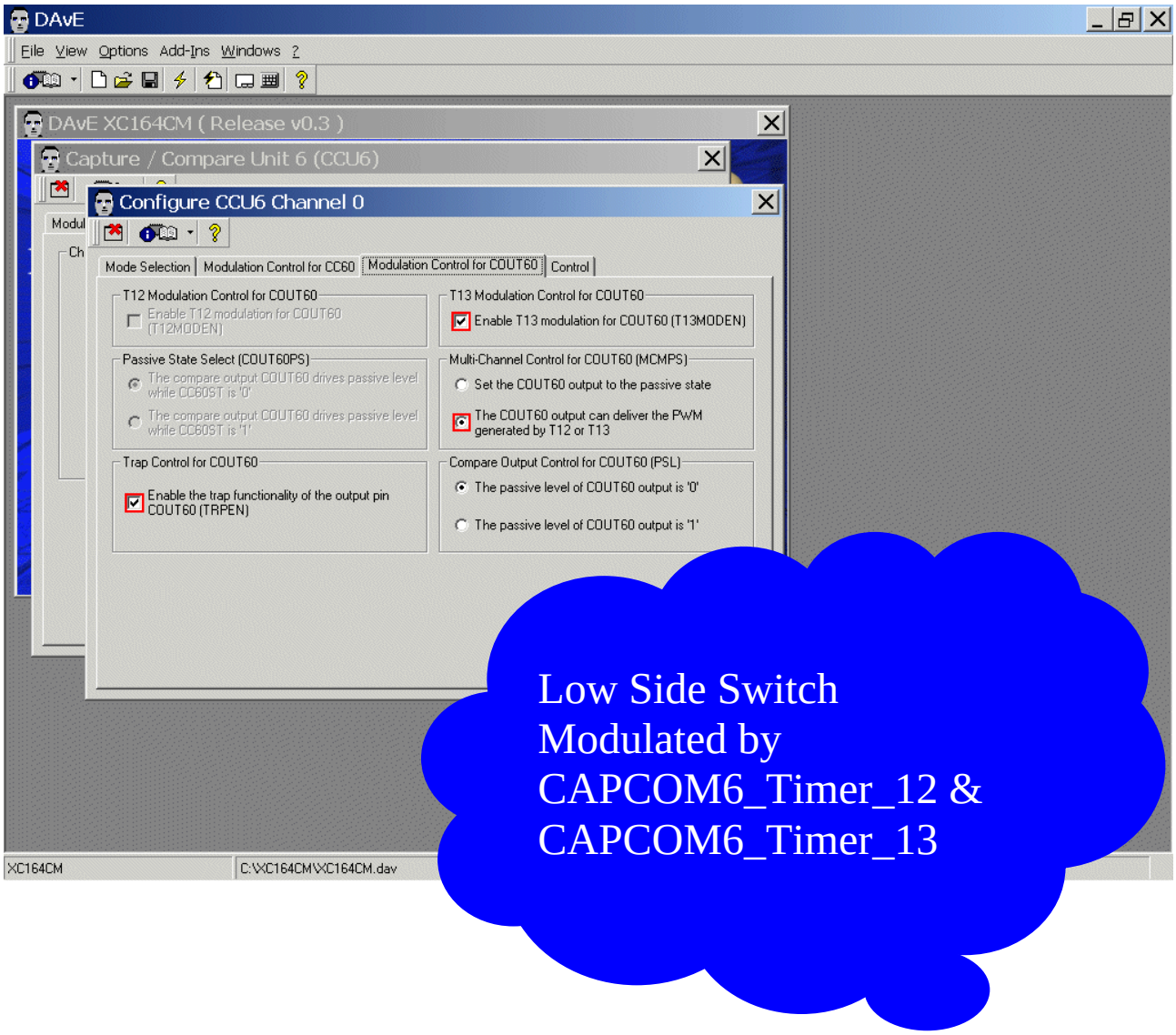


High Side Switch
Modulated by
CAPCOM6_Timer_12

CCU6: Channels: Configure Channel 0: Modulation Control for COUT60:
Trap Control for COUT60: click ✓ Enable the trap functionality of the output pin COUT60

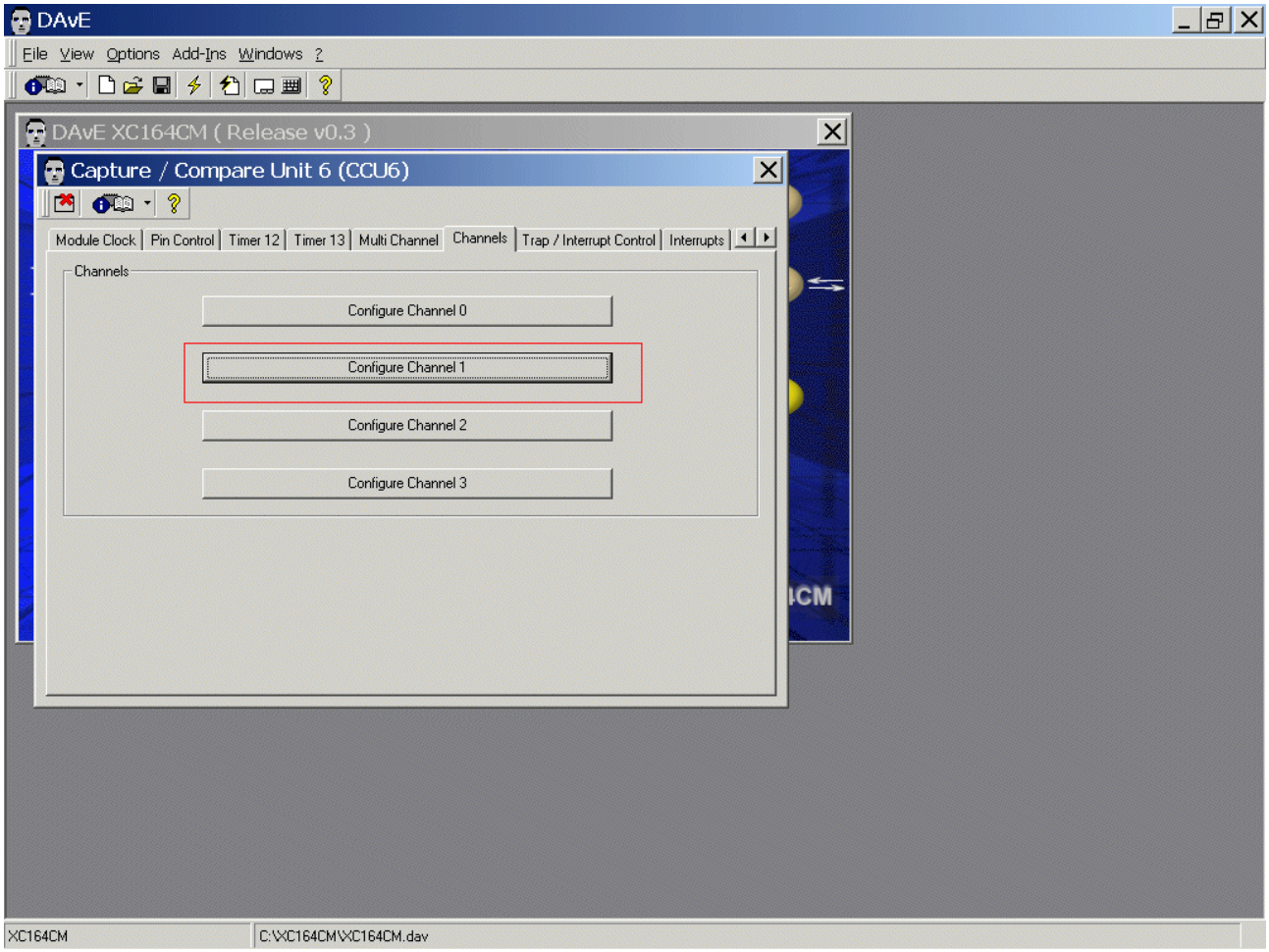
CCU6: Channels: Configure Channel 0: Modulation Control for COUT60:
T13 Modulation Control for COUT60: click ✓ Enable T13 modulation for COUT60

CCU6: Channels: Configure Channel 0: Modulation Control for COUT60:
Multi-Channel Control for COUT60: click ✓ The COUT60 output can deliver the PWM generated by T12 or T13

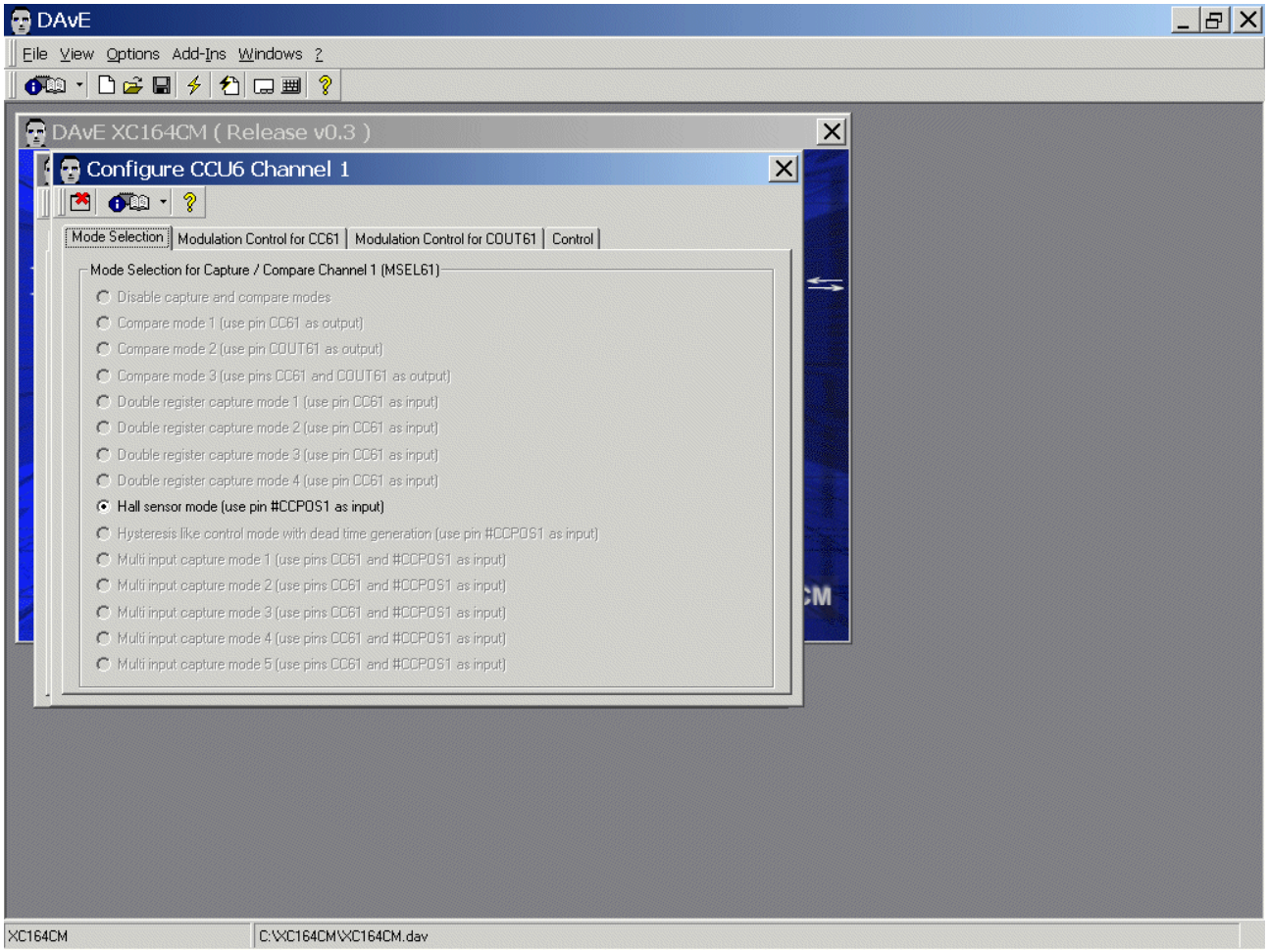


Exit this dialog now by clicking  the close button.

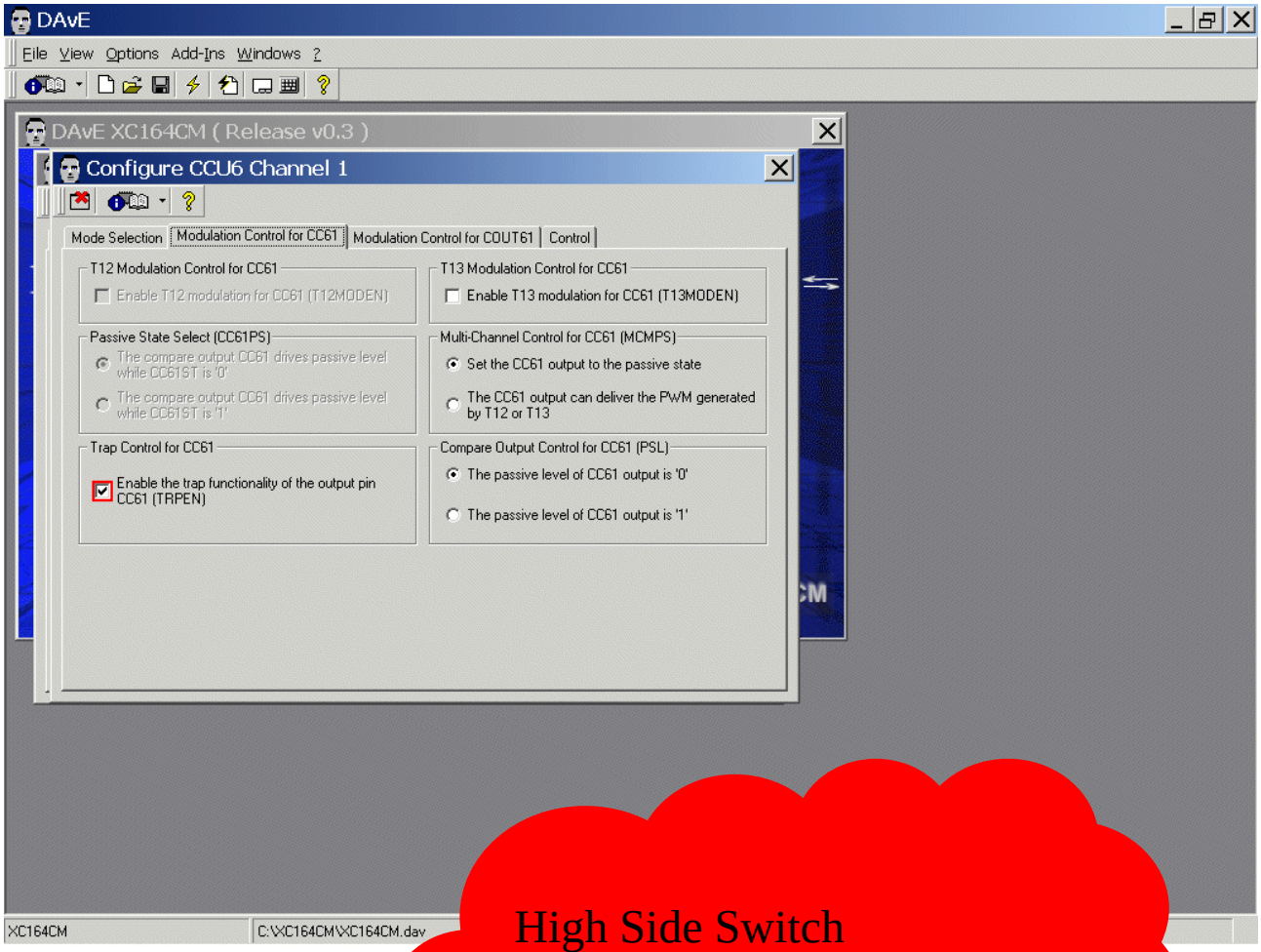
CCU6: Channels: click Configure Channel 1:



CCU6: Channels: Configure Channel 1:
Mode Selection: Mode Selection for Capture / Compare Channel 1: check Hall sensor mode



CCU6: Channels: Configure Channel 1: Modulation Control for CC61:
Trap Control for CC61: click ✓ Enable the trap functionality of the output pin CC61
(Enable the TRAP functionality for emergency shut down)

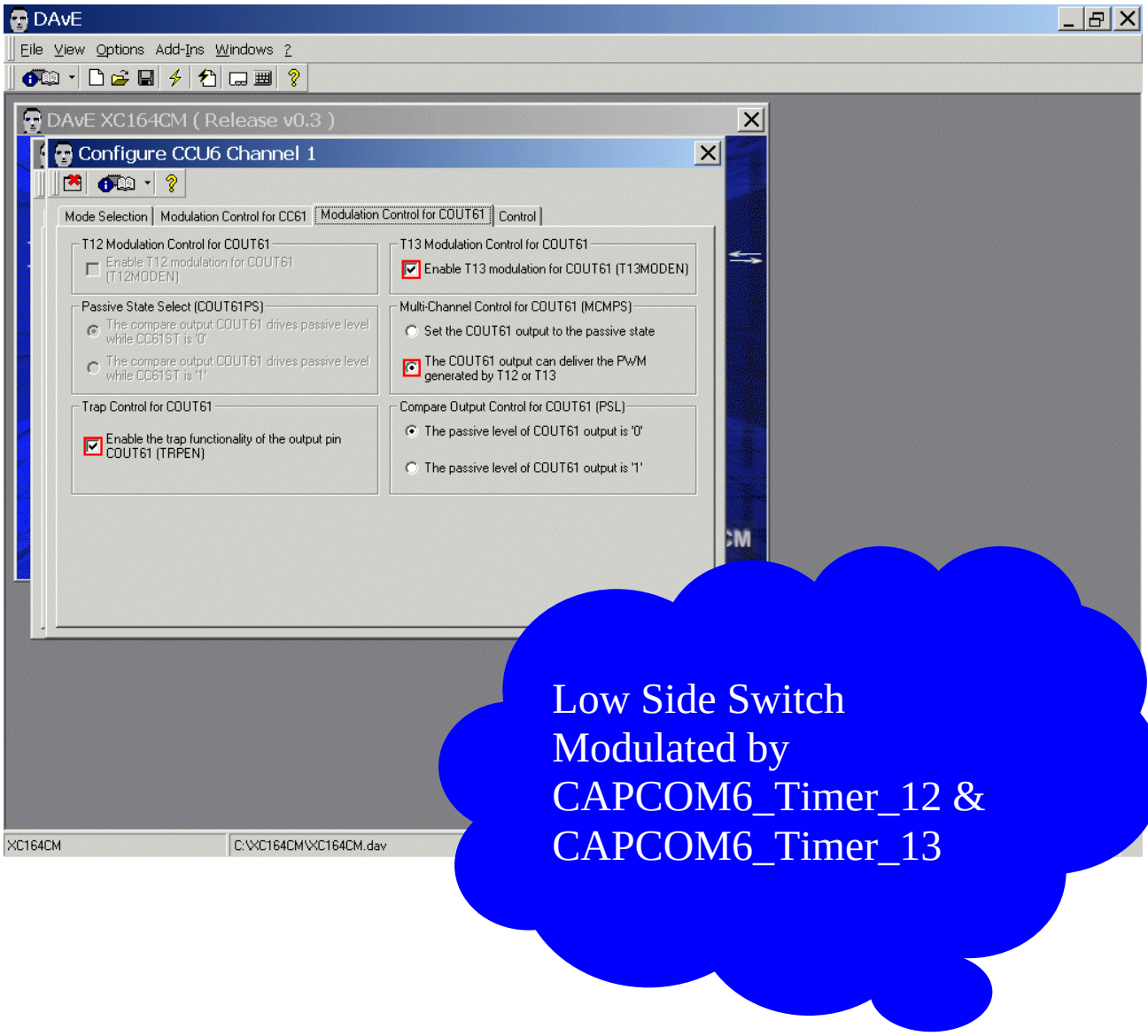


High Side Switch
Modulated by
CAPCOM6_Timer_12

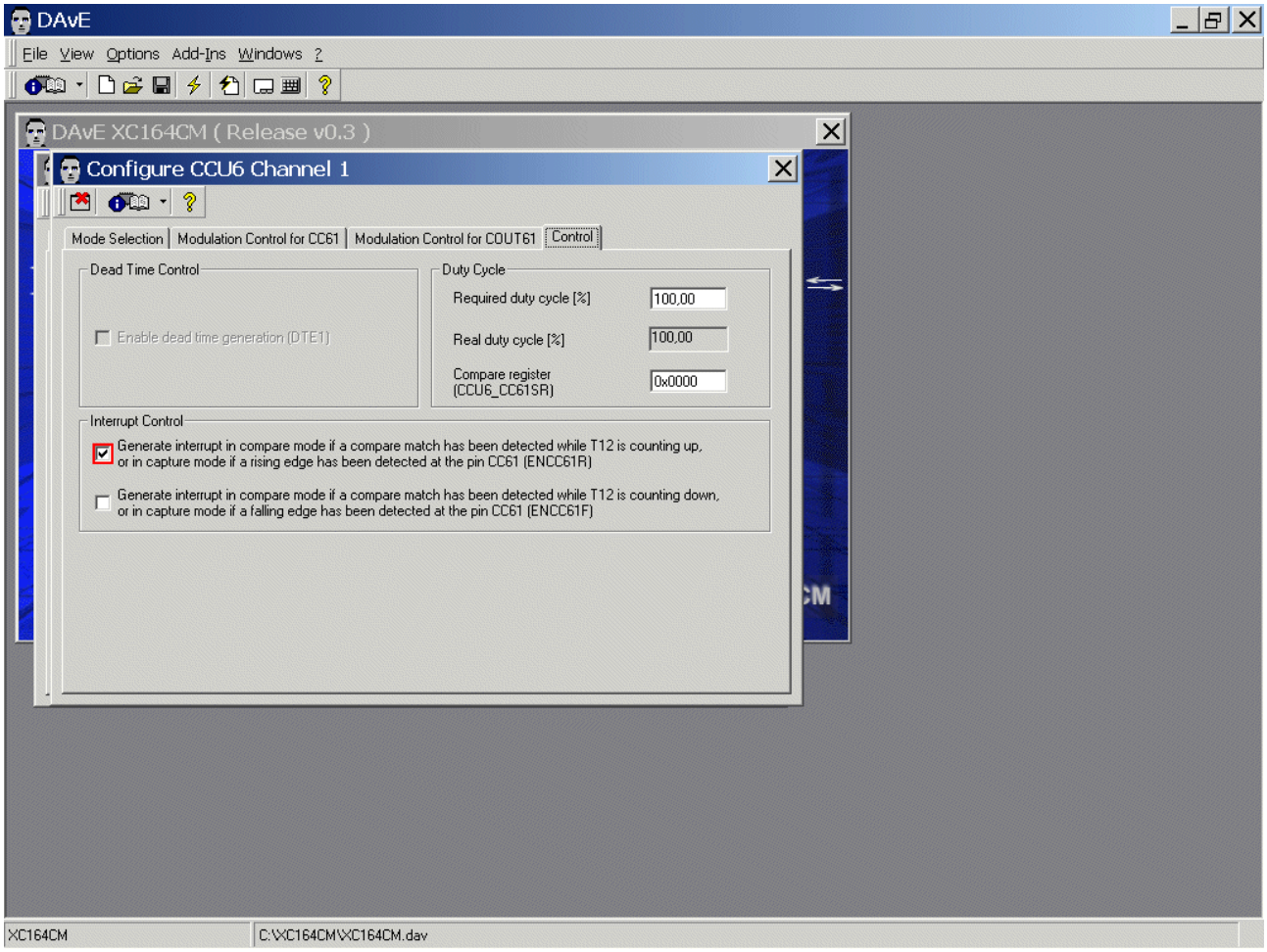
CCU6: Channels: Configure Channel 1: Modulation Control for COUT61:
Trap Control for COUT61: click ✓ Enable the trap functionality of the output pin COUT61

CCU6: Channels: Configure Channel 1: Modulation Control for COUT61:
T13 Modulation Control for COUT61: click ✓ Enable T13 modulation for COUT61

CCU6: Channels: Configure Channel 1: Modulation Control for COUT61:
Multi-Channel Control for COUT61: click ✓ The COUT61 output can deliver the PWM generated by T12 or T13

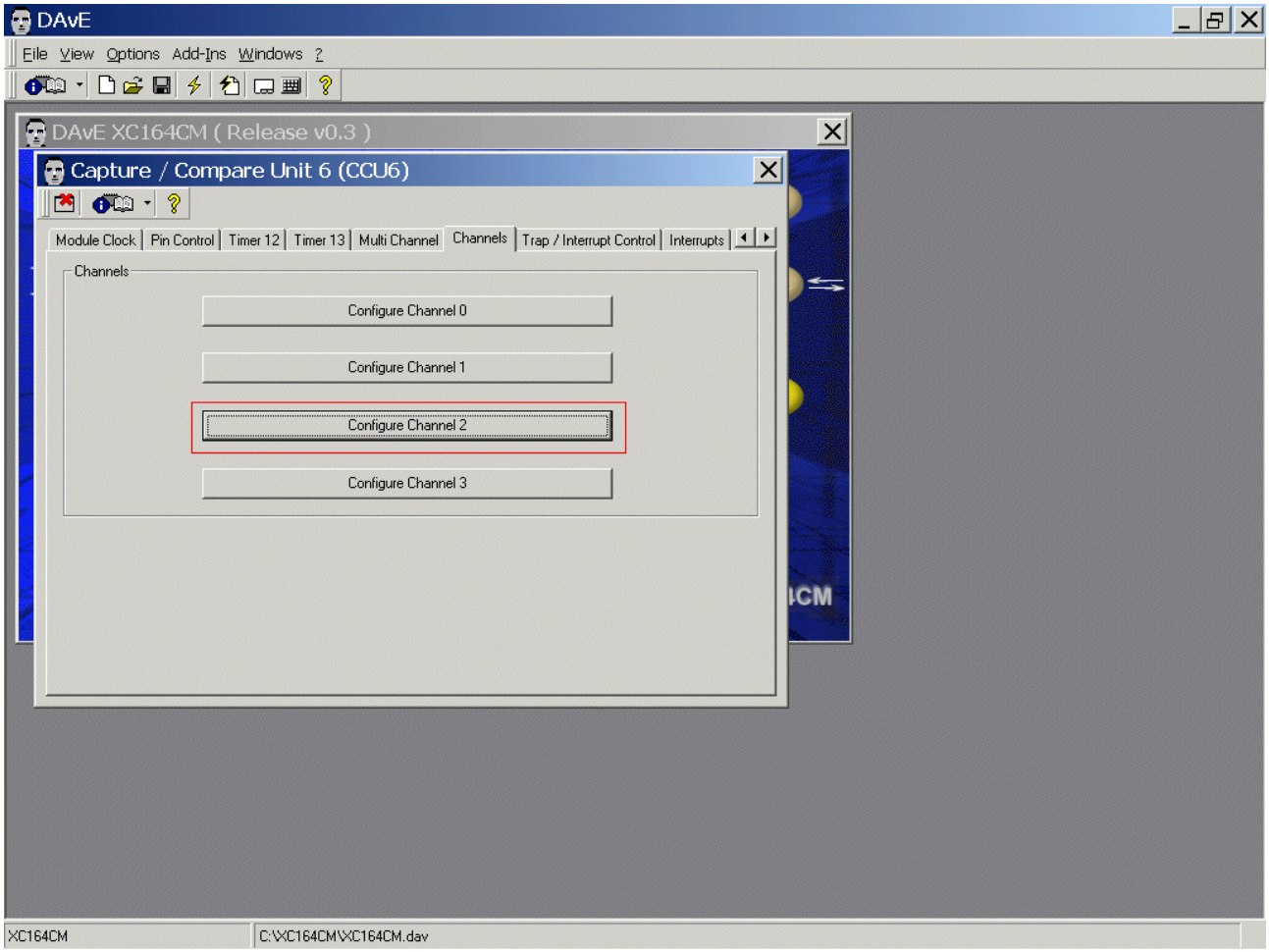


CCU6: Channels: Configure Channel 1: Control:
Interrupt Control: click ✓ Generate interrupt in compare mode if a compare match has been detected while T12 is counting up, or in capture mode ….

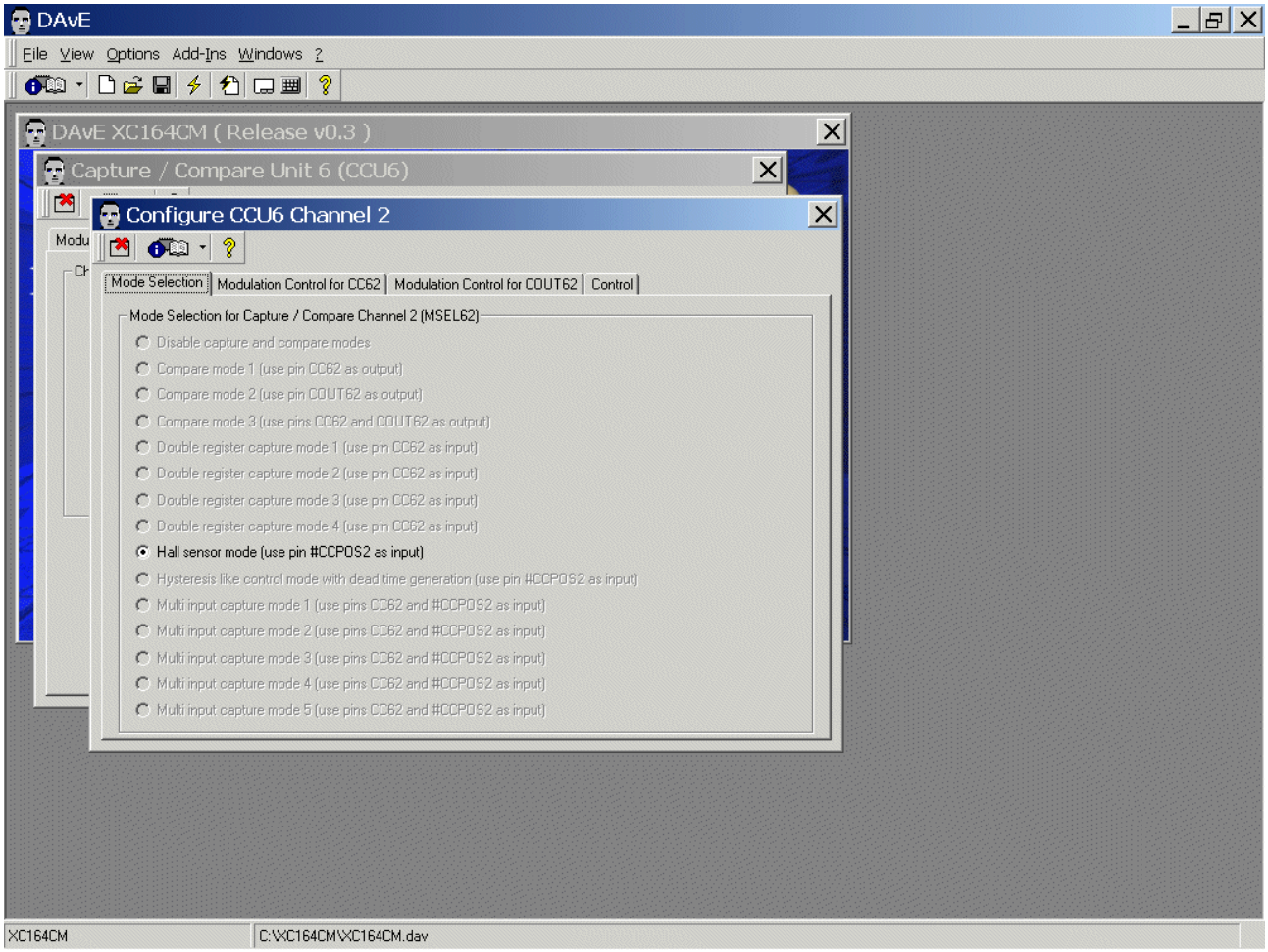


Exit this dialog now by clicking  the close button.

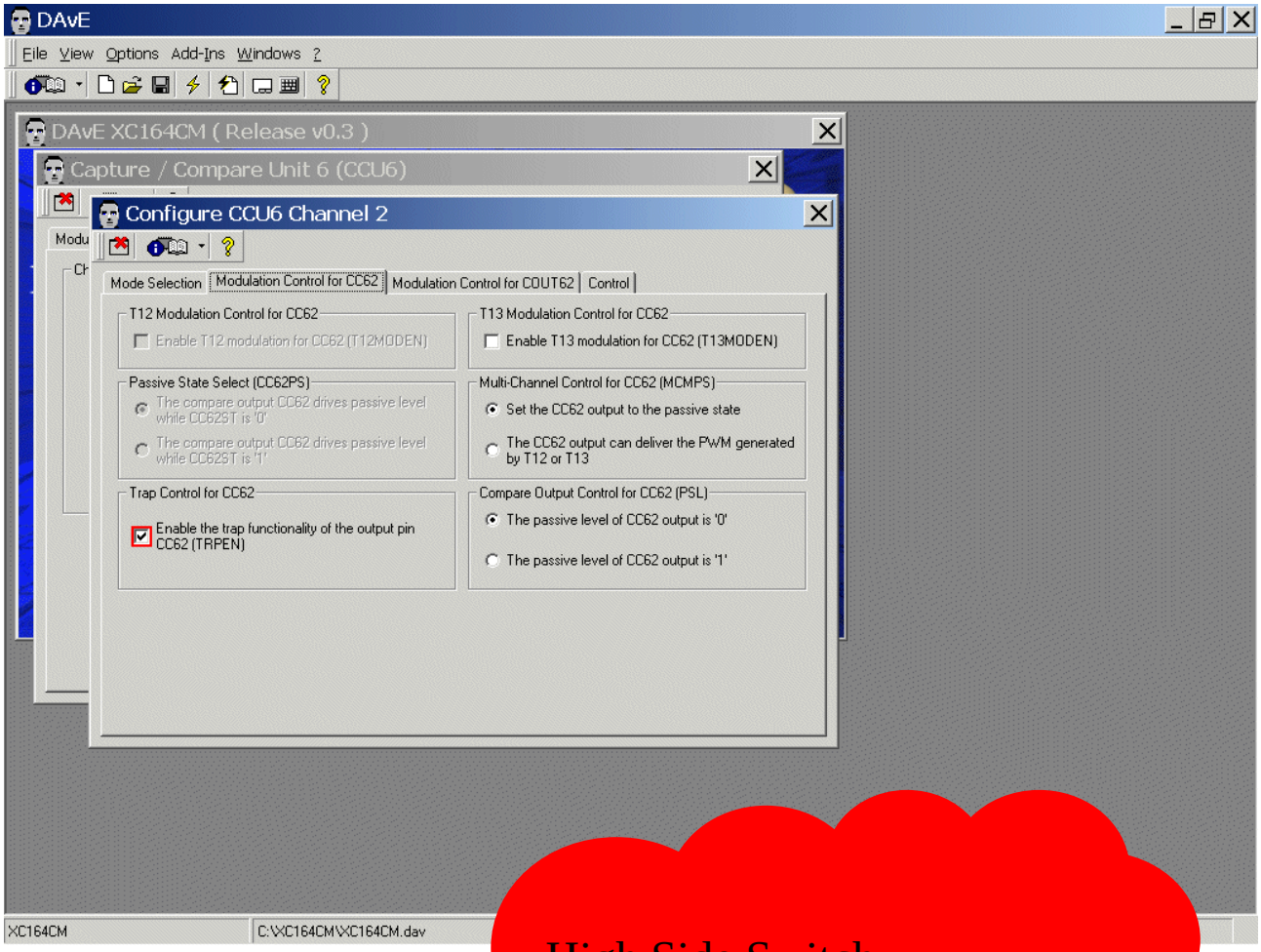
CCU6: Channels: click Configure Channel 2:



CCU6: Channels: Configure Channel 2:
Mode Selection: Mode Selection for Capture / Compare Channel 2: check Hall sensor mode



CCU6: Channels: Configure Channel 2: Modulation Control for CC62:
Trap Control for CC62: click ✓ Enable the trap functionality of the output pin CC62
(Enable the TRAP functionality for emergency shut down)

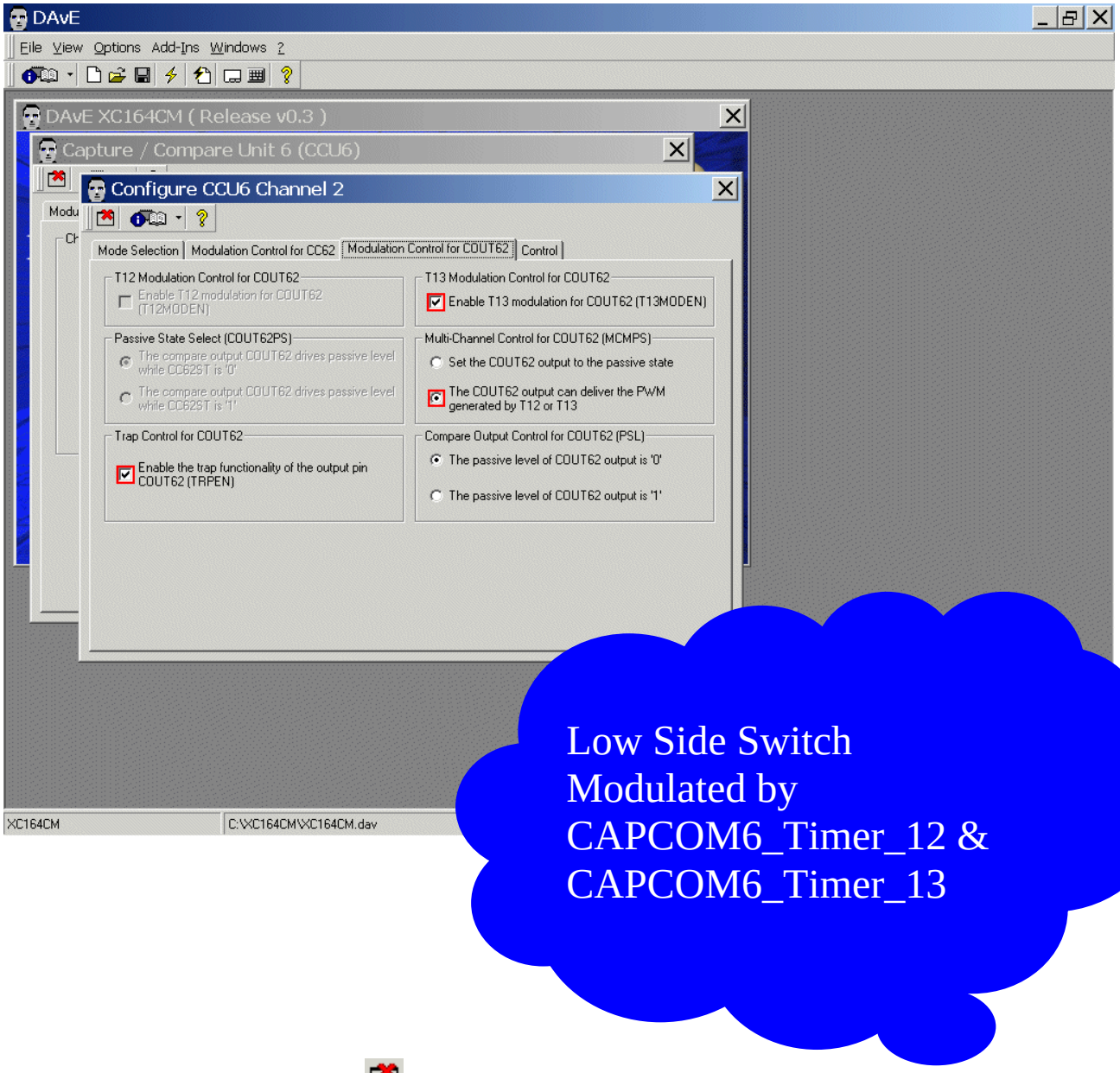


High Side Switch
Modulated by
CAPCOM6_Timer_12

CCU6: Channels: Configure Channel 2: Modulation Control for COUT62:
Trap Control for COUT62: click ✓ Enable the trap functionality of the output pin COUT62

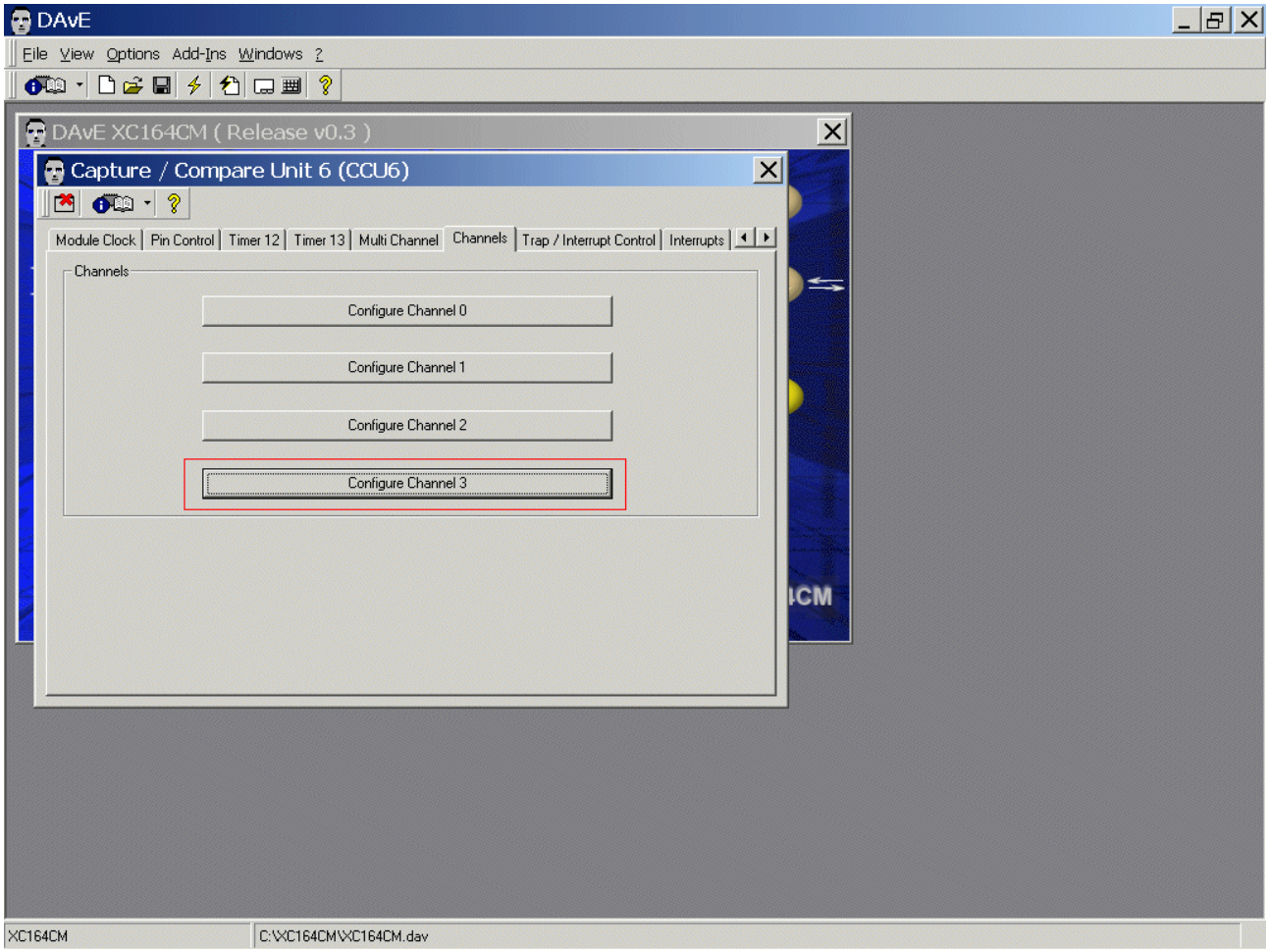
CCU6: Channels: Configure Channel 2: Modulation Control for COUT62:
T13 Modulation Control for COUT62: click ✓ Enable T13 modulation for COUT62

CCU6: Channels: Configure Channel 2: Modulation Control for COUT62:
Multi-Channel Control for COUT62: click ✓ The COUT62 output can deliver the PWM generated by T12 or T13



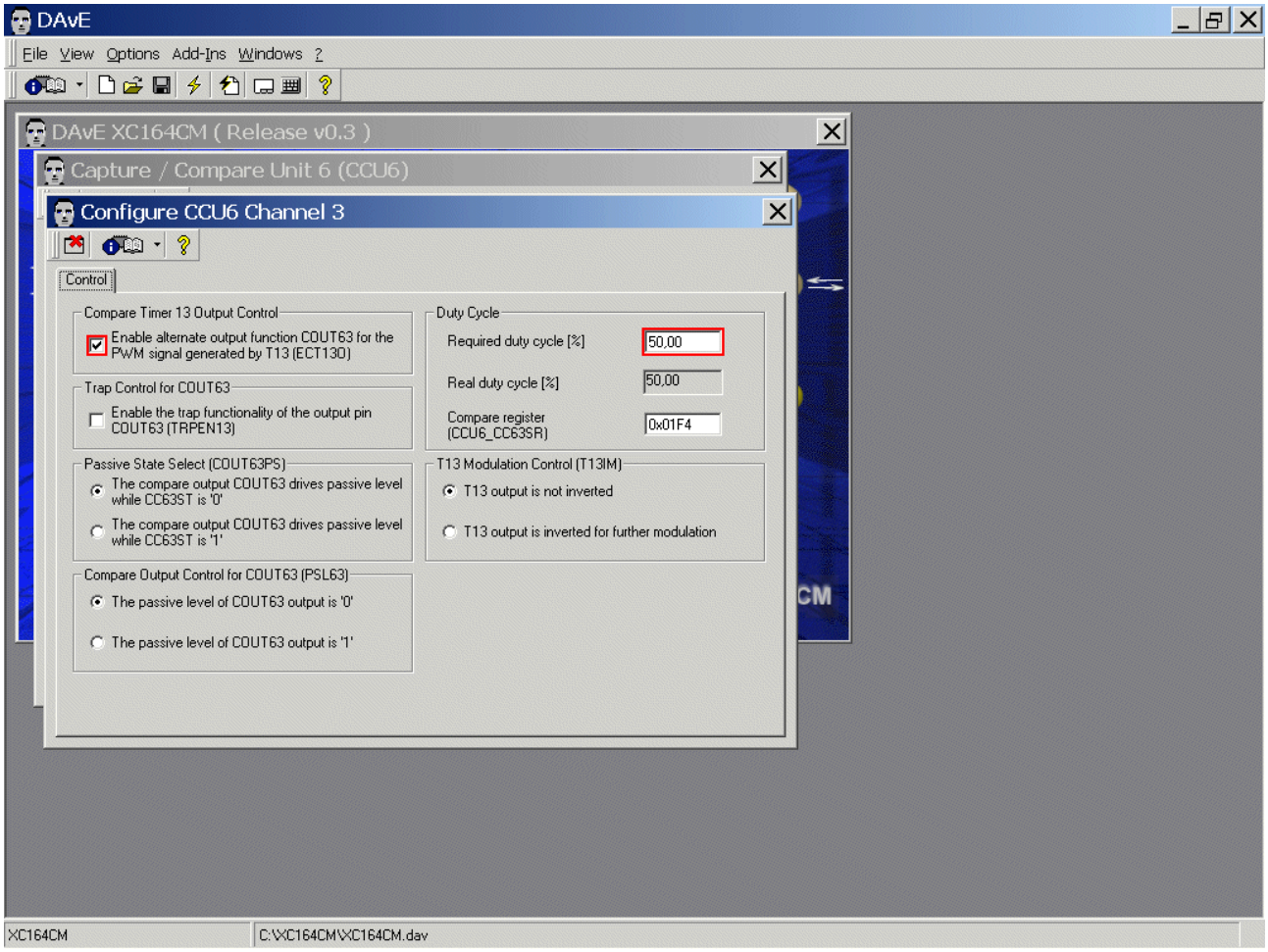
Exit this dialog now by clicking the close button.

CCU6: Channels: click Configure Channel 3:



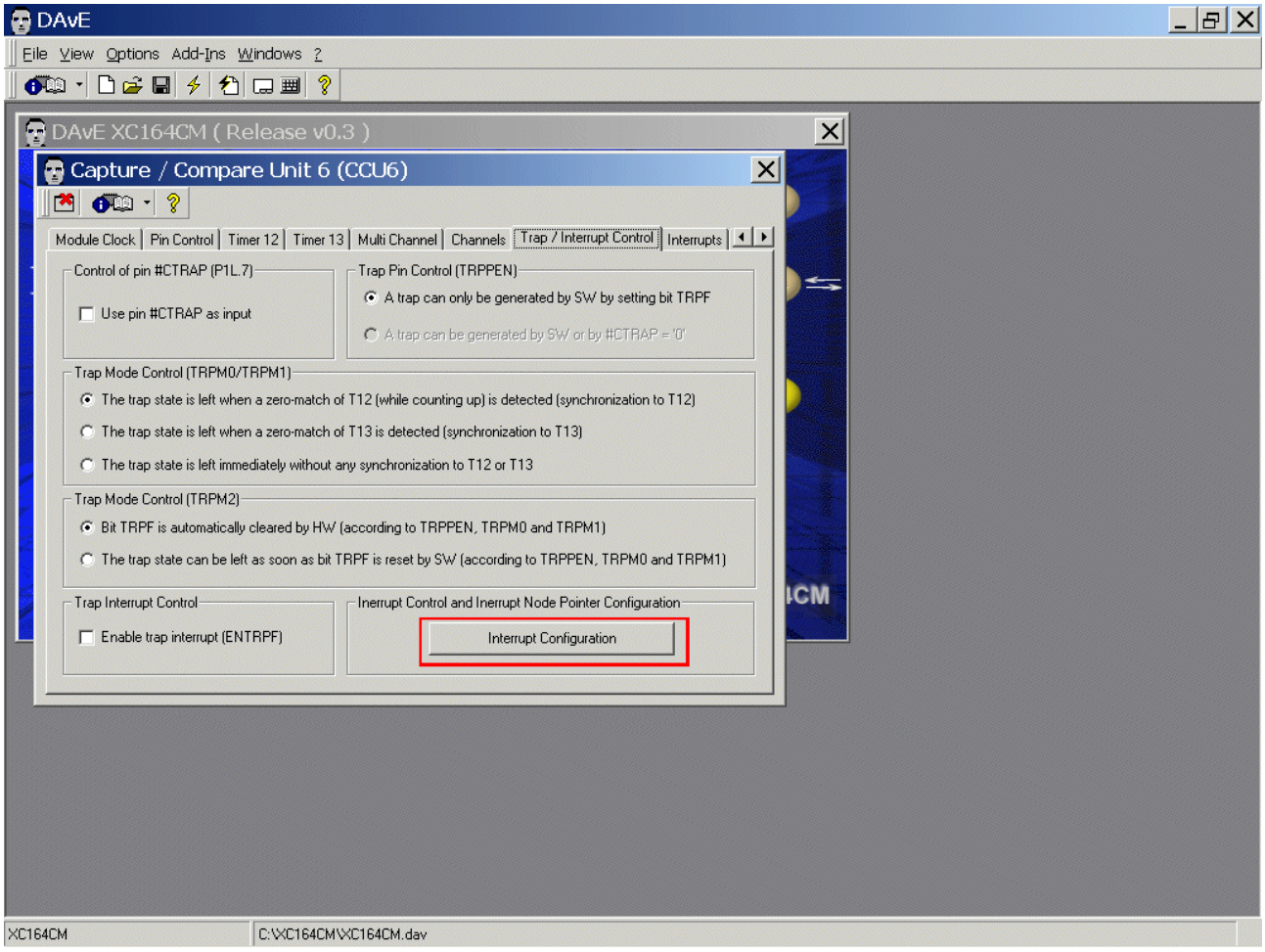
CCU6: Channels: Configure Channel 3:
Control: Compare Timer 13 Output Control: click ✓ Enable alternate output function COUT63 for the PWM signal generated by T13

CCU6: Channels: Configure Channel 3:
Duty Cycle: Required duty cycle [%]: insert 50 <ENTER>

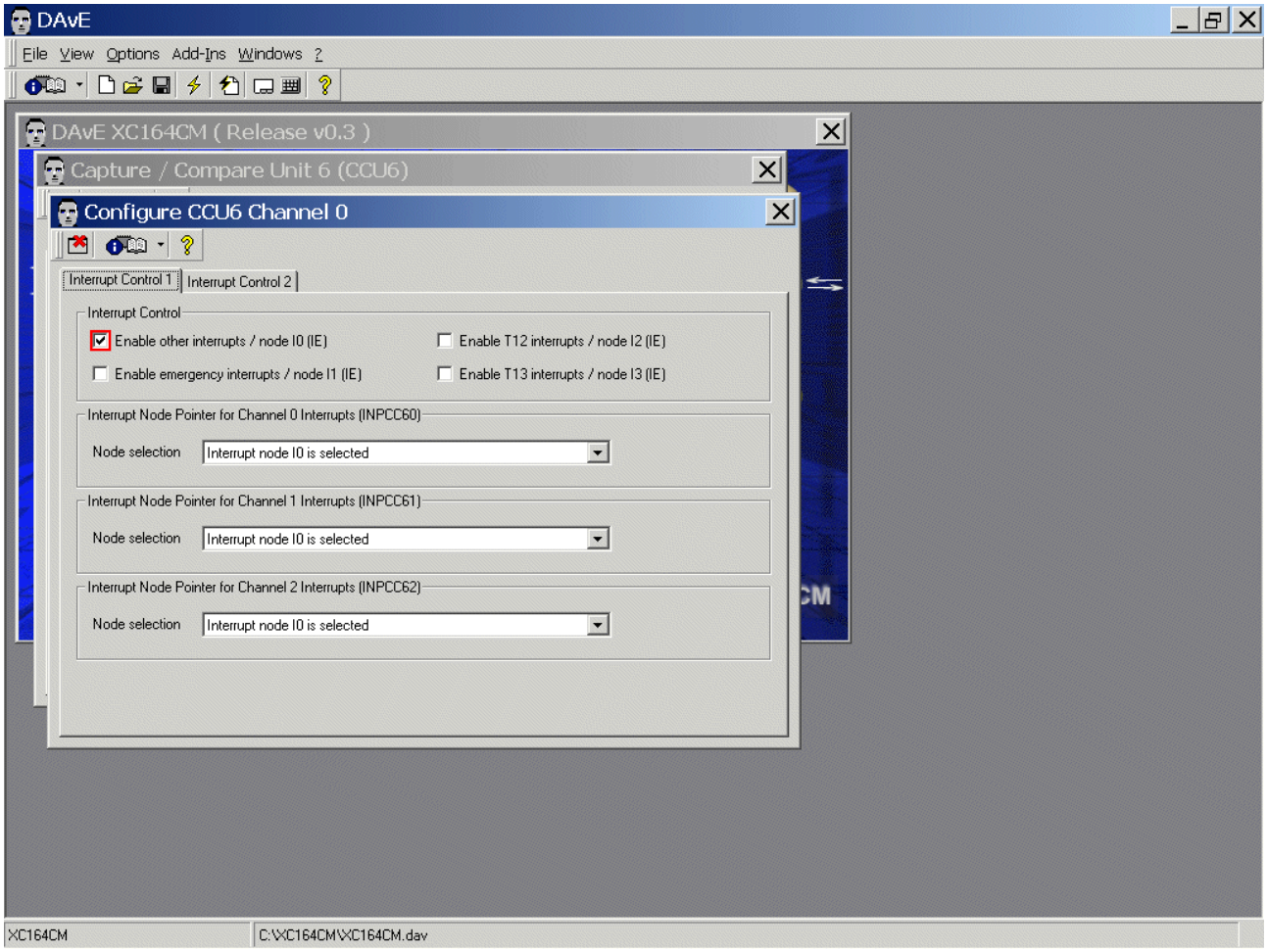


Exit this dialog now by clicking  the close button.

CCU6: Trap/Interrupt Control: click Interrupt Configuration

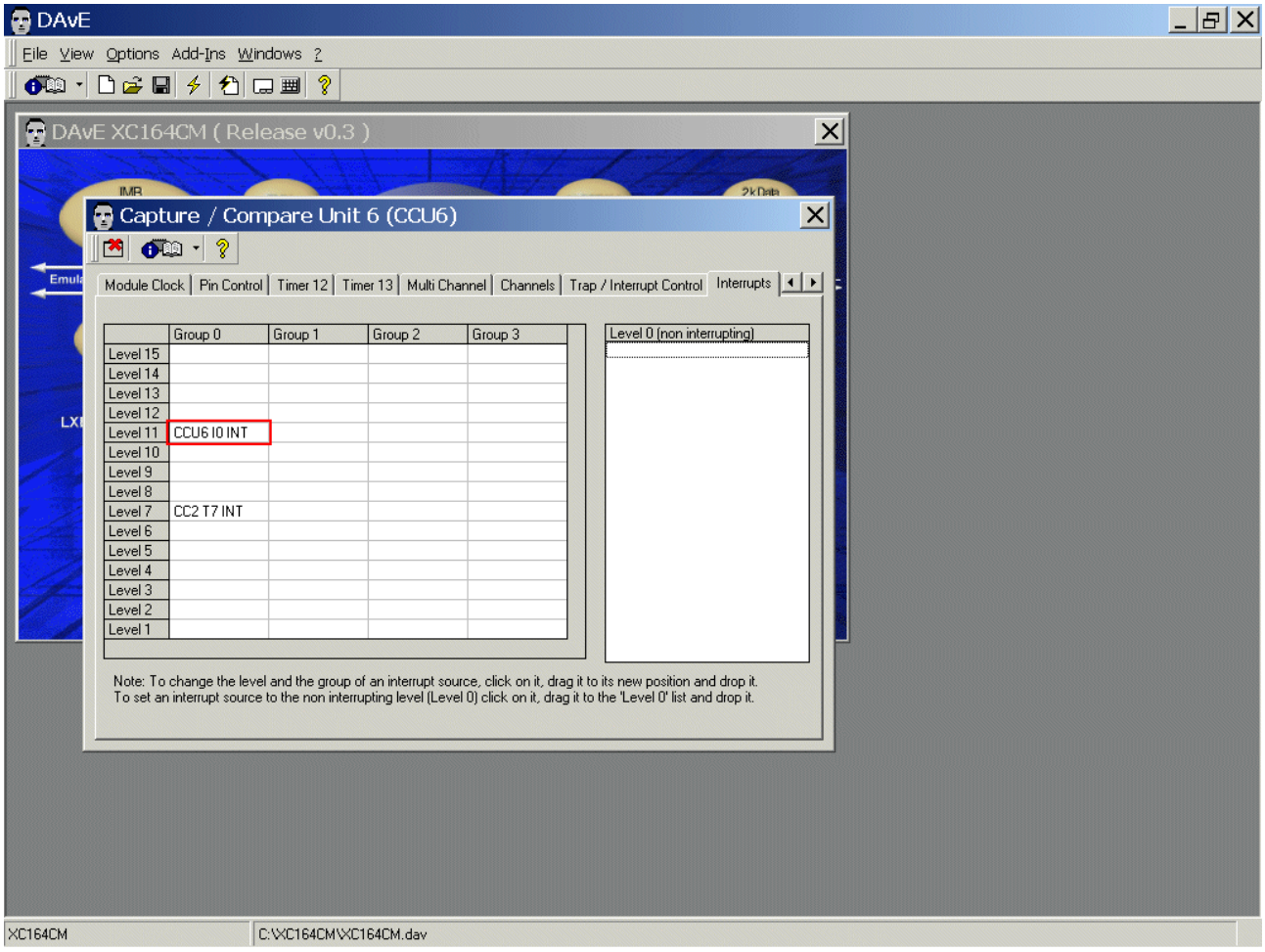


CCU6: Trap/Interrupt Control: click Interrupt Configuration
Interrupt Control 1: Interrupt Control: click ✓ Enable other interrupts / node I0
(Enable other Interrupts for Hall Sensor)

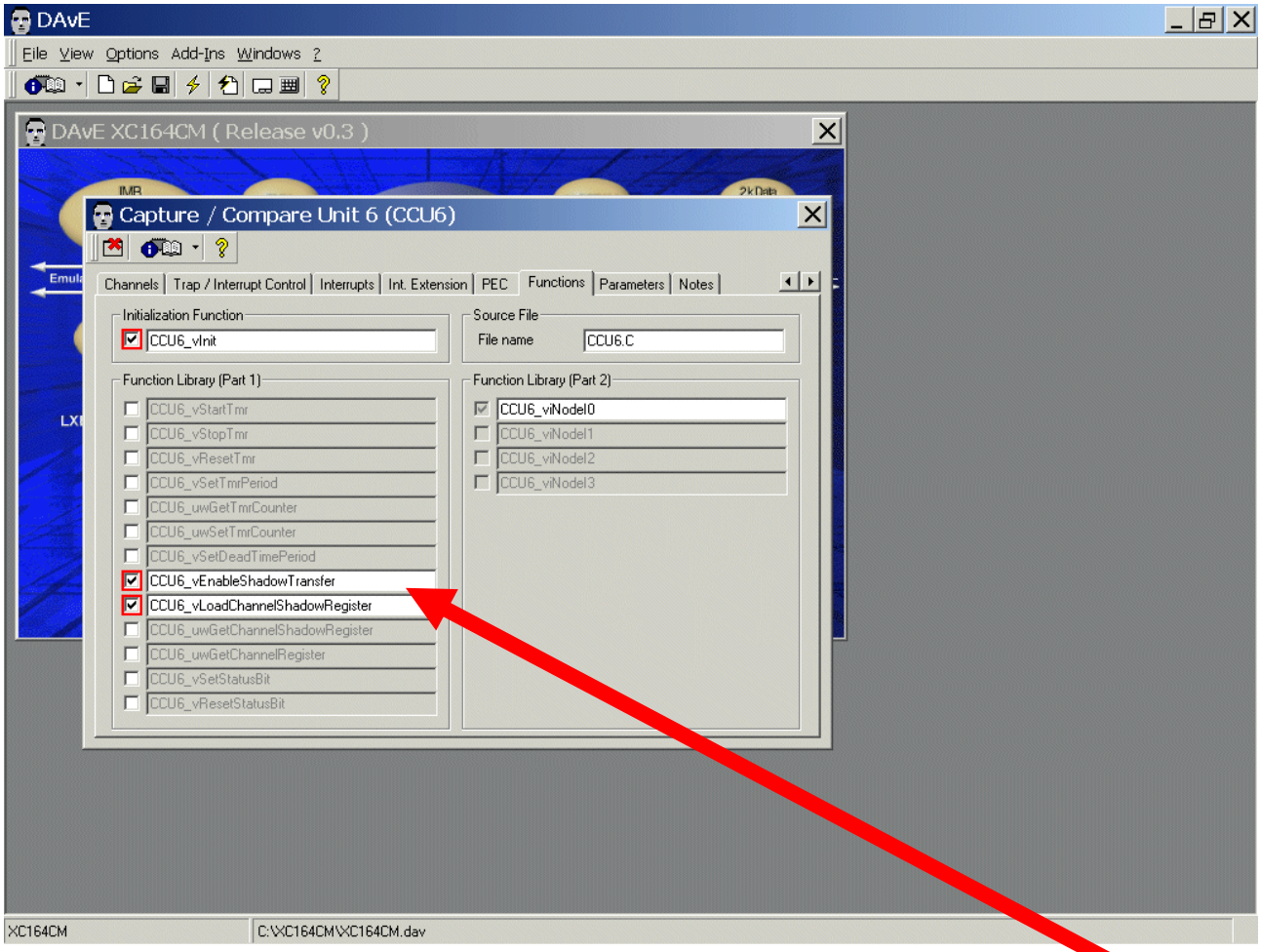


Exit this dialog now by clicking  the close button.

CCU6: Interrupts: drag and drop CCU6 I0 INT to Level 11, Group 0



CCU6: Functions: Initialization Function: click CCU6_vInit
CCU6: Functions: Function Library (Part 1): click CCU6_vEnableShadowTransfer
CCU6: Functions: Function Library (Part 1): click CCU6_vLoadChannelShadowRegister

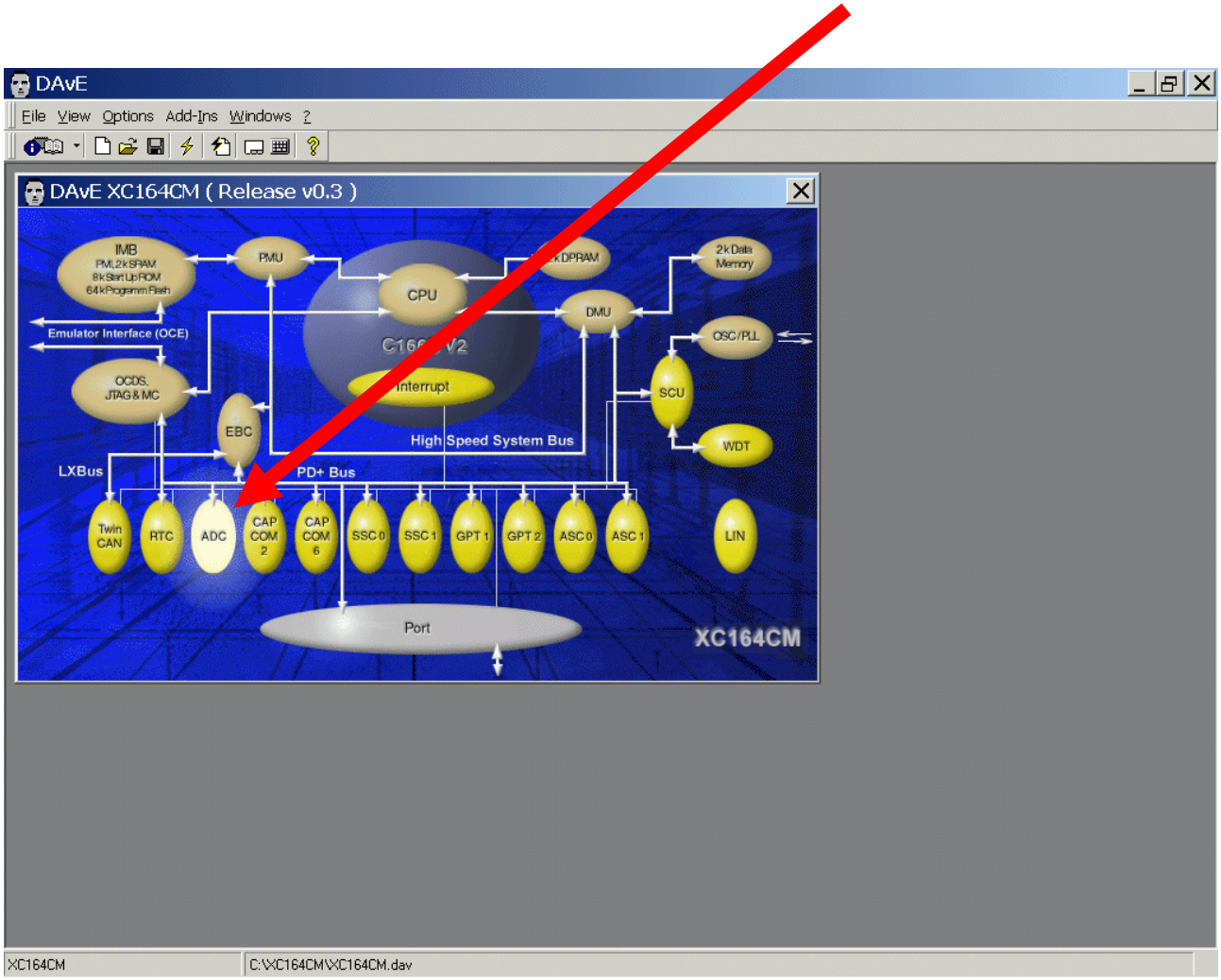


CCU6_vEnableShadowTransfer() and CCU6_vLoadChannelShadowRegister() will be used to Change The Speed Of The Motor (see Chapter 9 in this document) – by increasing/ decreasing the DutyCycle of CAPCOM6_Timer_13

Exit this dialog now by clicking  the close button.

Configuration of the ADC (configure as you can see in the screenshots):

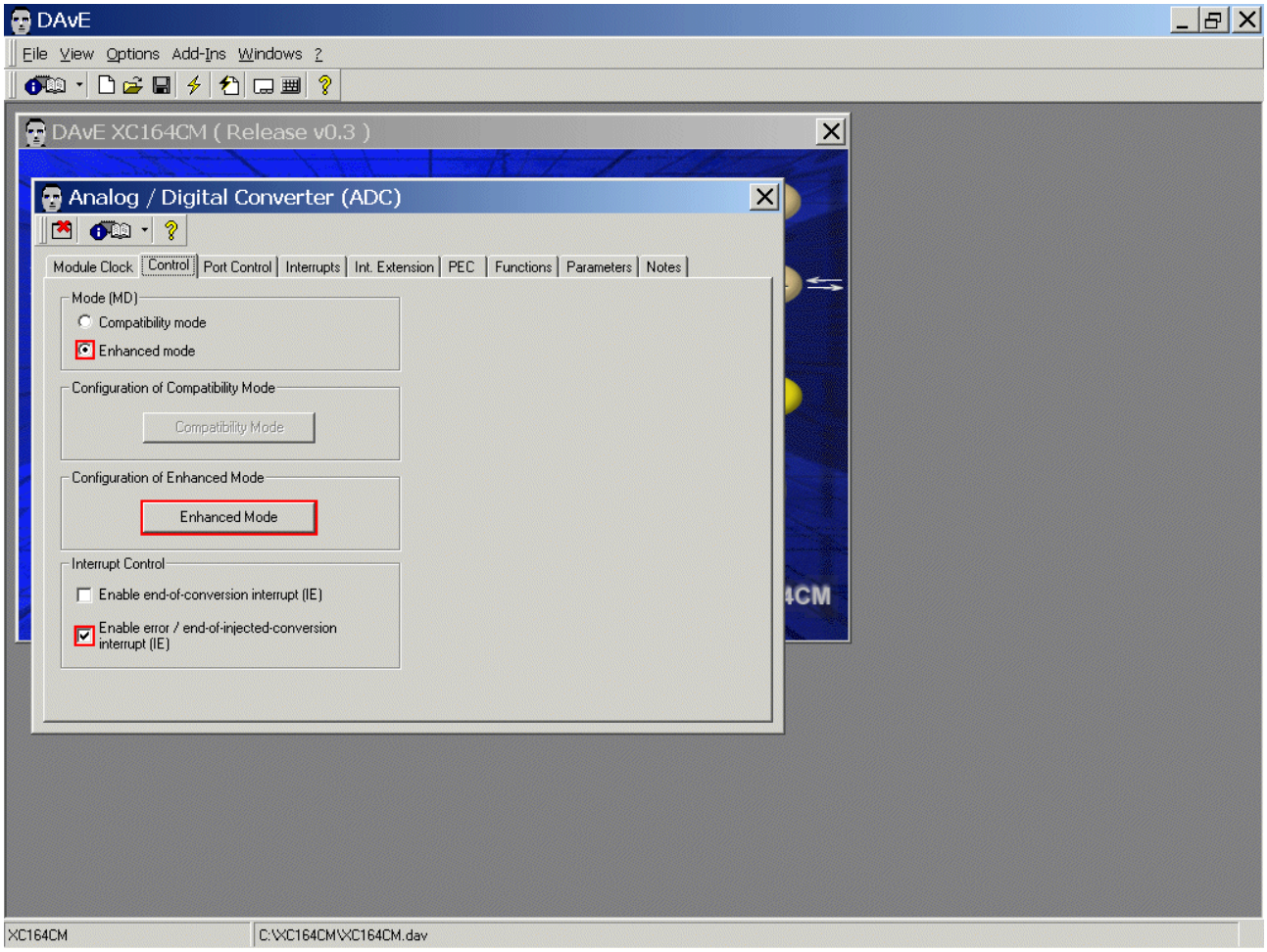
The configuration window can be opened by clicking the specific block/module.



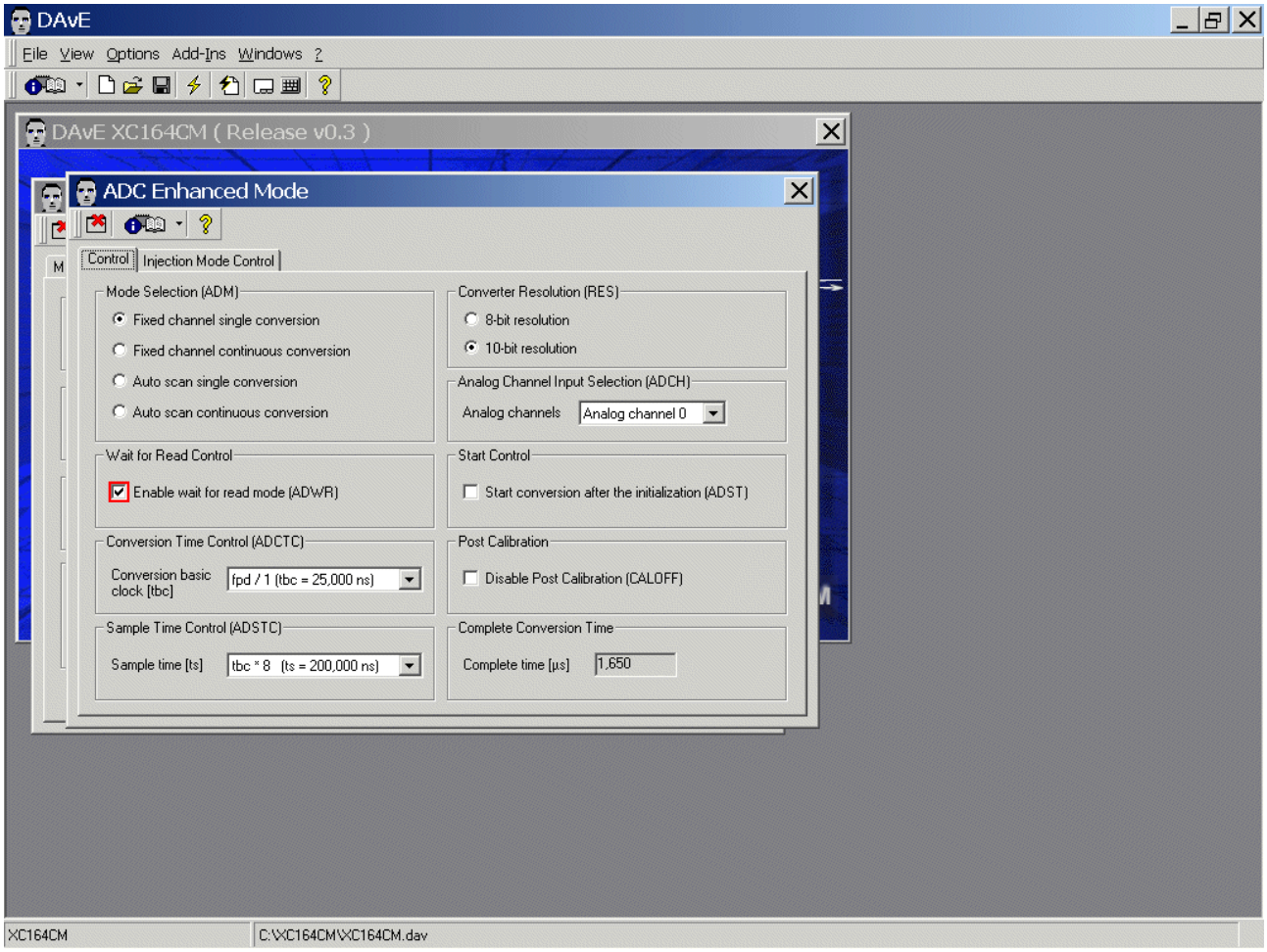
Note :

The ADC will be used for “DC-Link Shunt Current Measurement” (see Chapter 7 in this document).

ADC: Control: Mode: click ⊙ Enhanced mode
ADC: Control: Interrupt Control: click ✓ Enable error / end-of-injected-conversion interrupt
ADC: Control: Configuration of Enhanced Mode: click Enhanced Mode



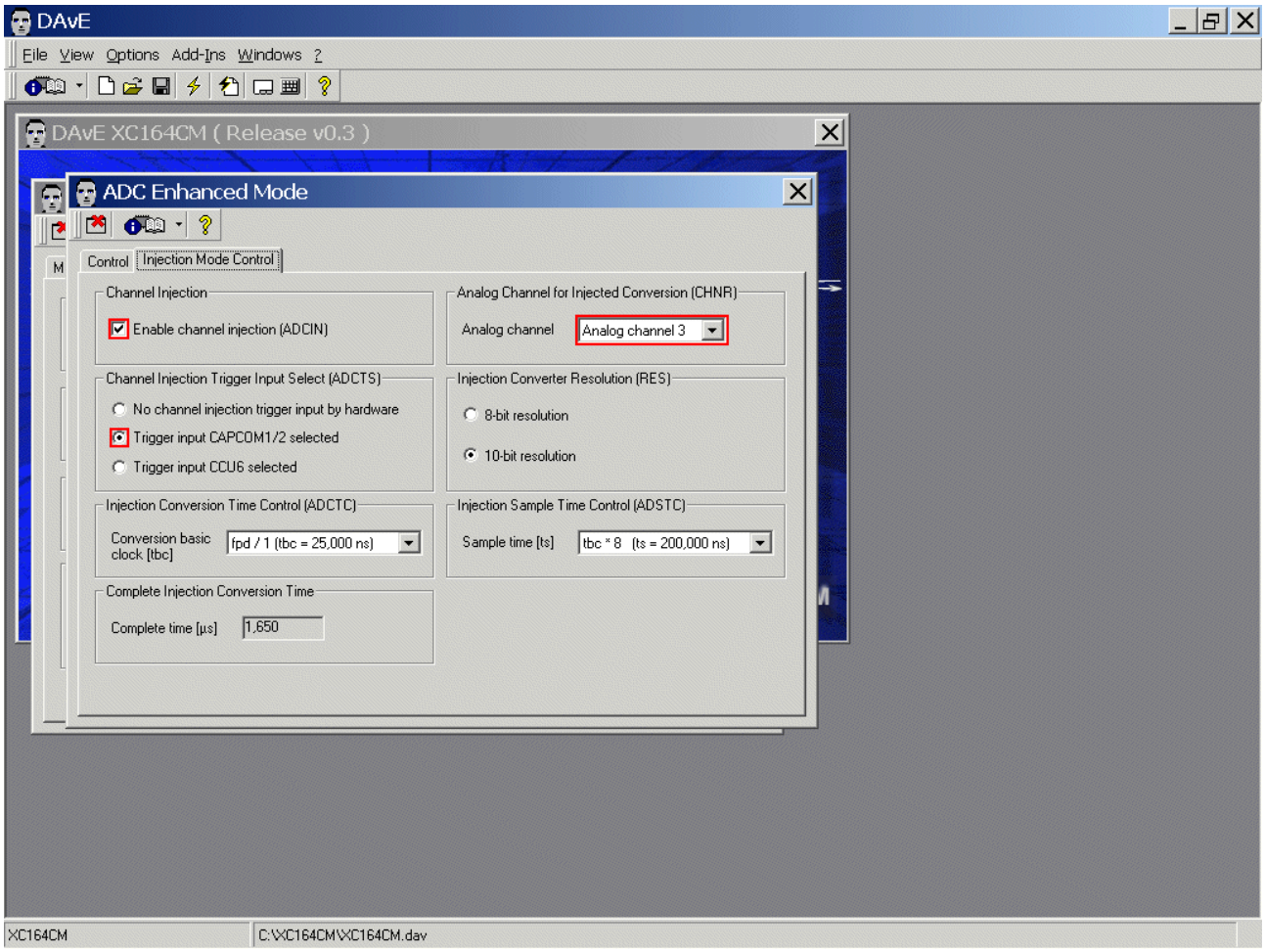
ADC: Control: Configuration of Enhanced Mode: click Enhanced Mode
Control: Wait for Read Control: click ✓ Enable wait for read mode



ADC: Control: Configuration of Enhanced Mode: click Enhanced Mode
Injection Mode Control: Channel Injection: click ✓ Enable channel injection

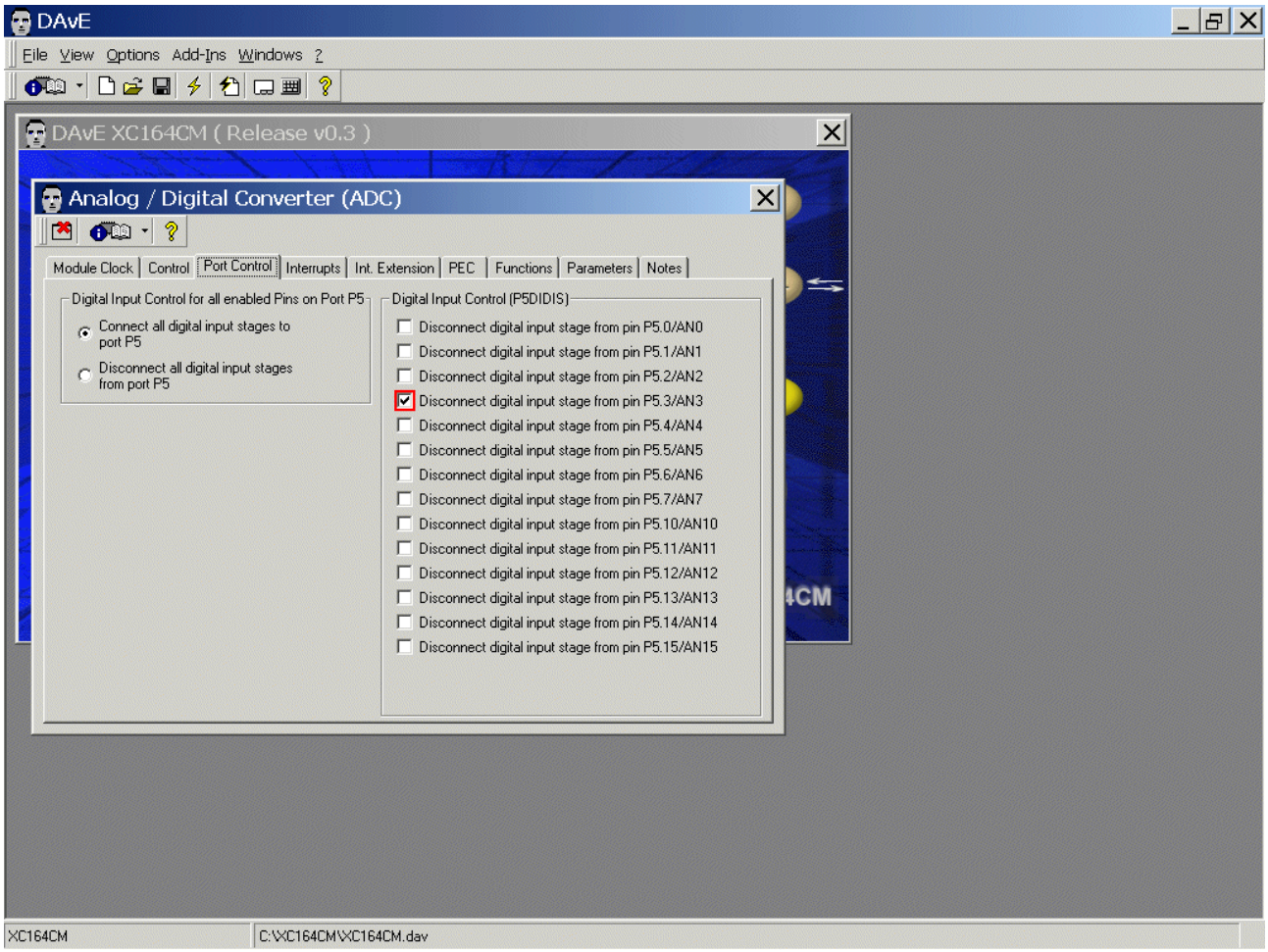
ADC: Control: Configuration of Enhanced Mode: click Enhanced Mode
Injection Mode Control: Channel Injection Trigger Input Select: click/check ⊙ Trigger input CAPCOM1/2 selected

ADC: Control: Configuration of Enhanced Mode: click Enhanced Mode
Injection Mode Control: Analog Channel for Injected Conversion: Analog channel: select Analog channel 3

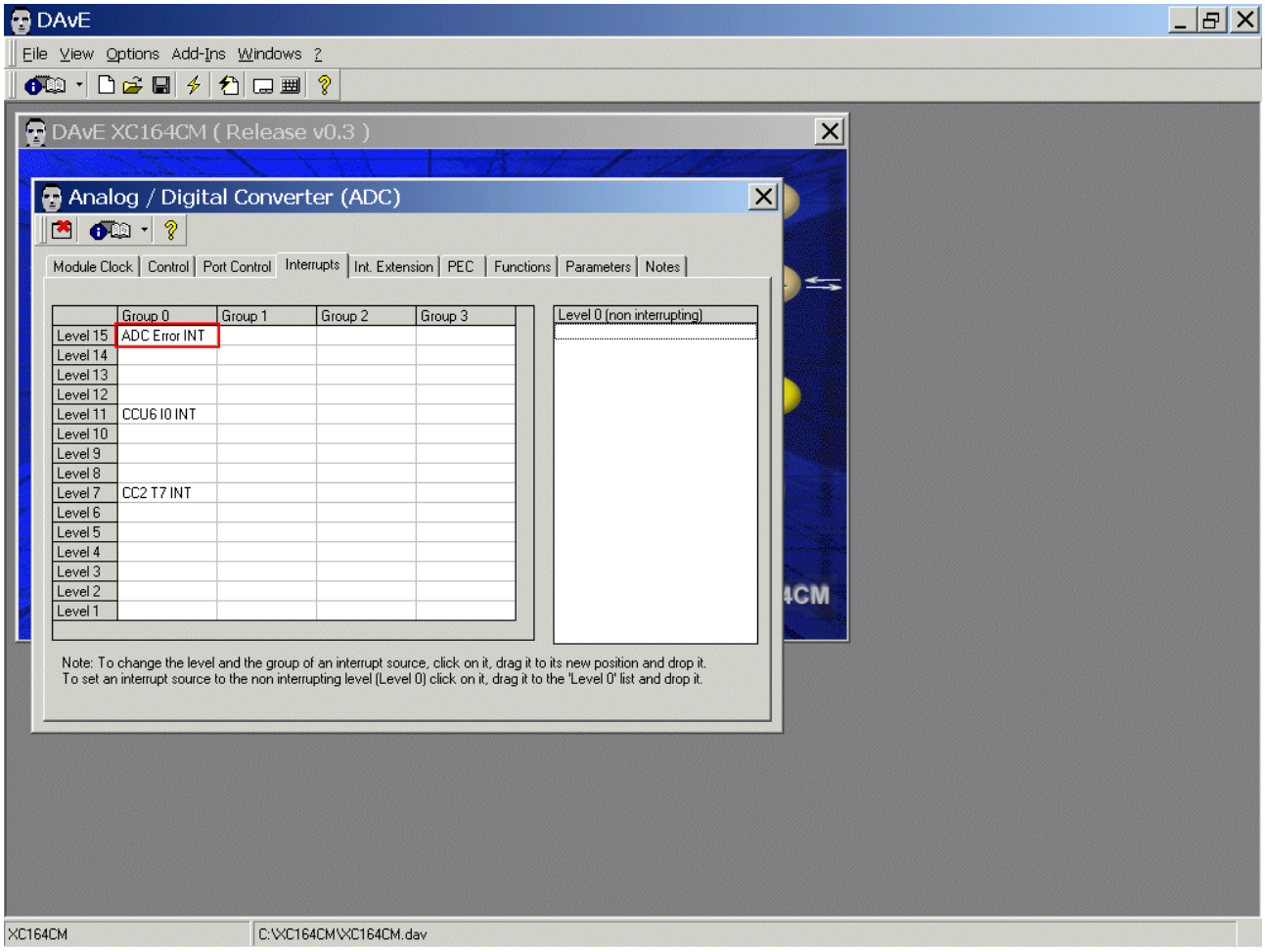


Exit this dialog now by clicking  the close button.

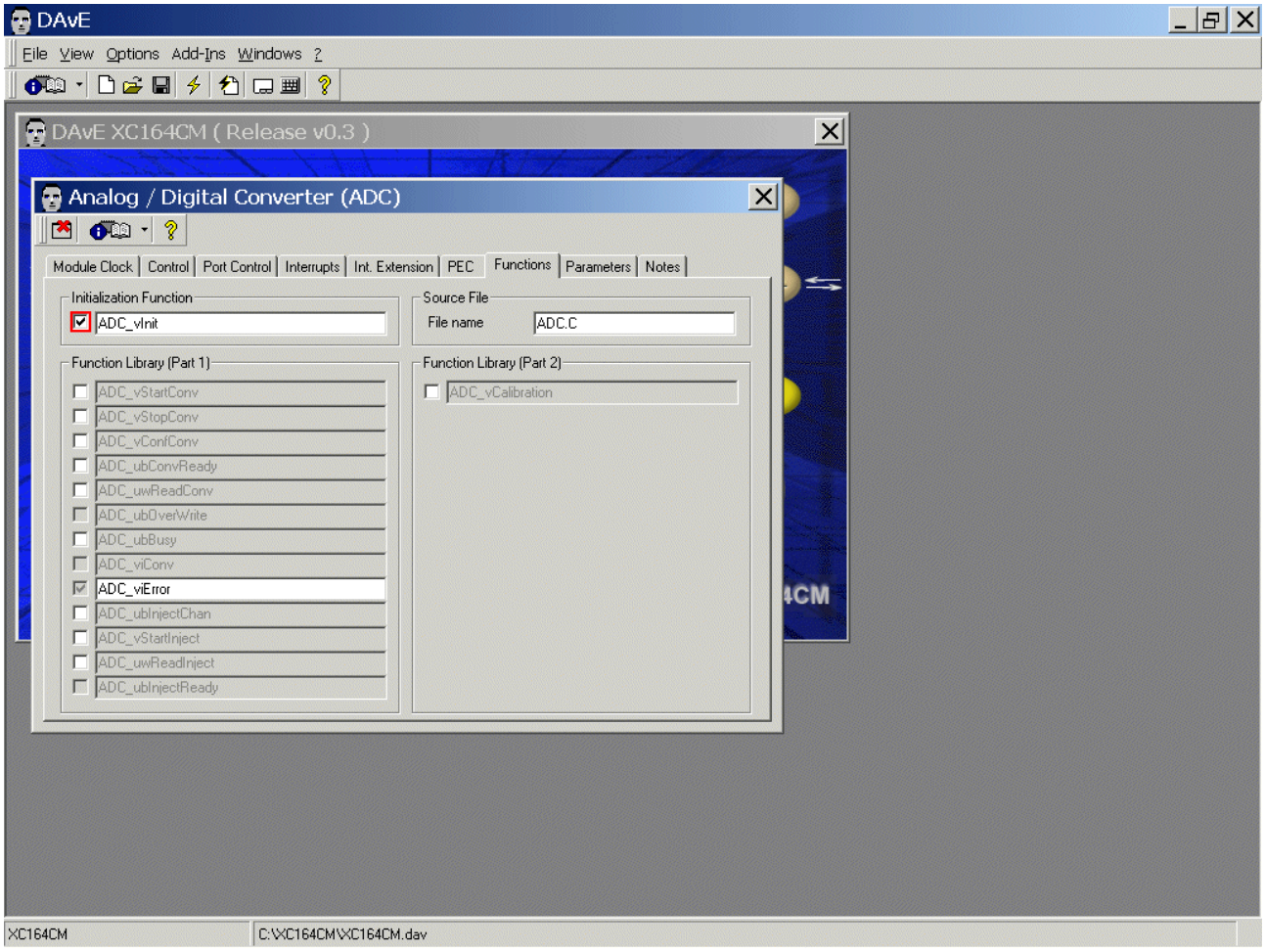
ADC: Port Control: Digital Input Control: click ✓ Disconnect digital input stage from pin P5.3/AN3



ADC: Interrupts: drag and drop ADC Error INT to Level 15, Group 0



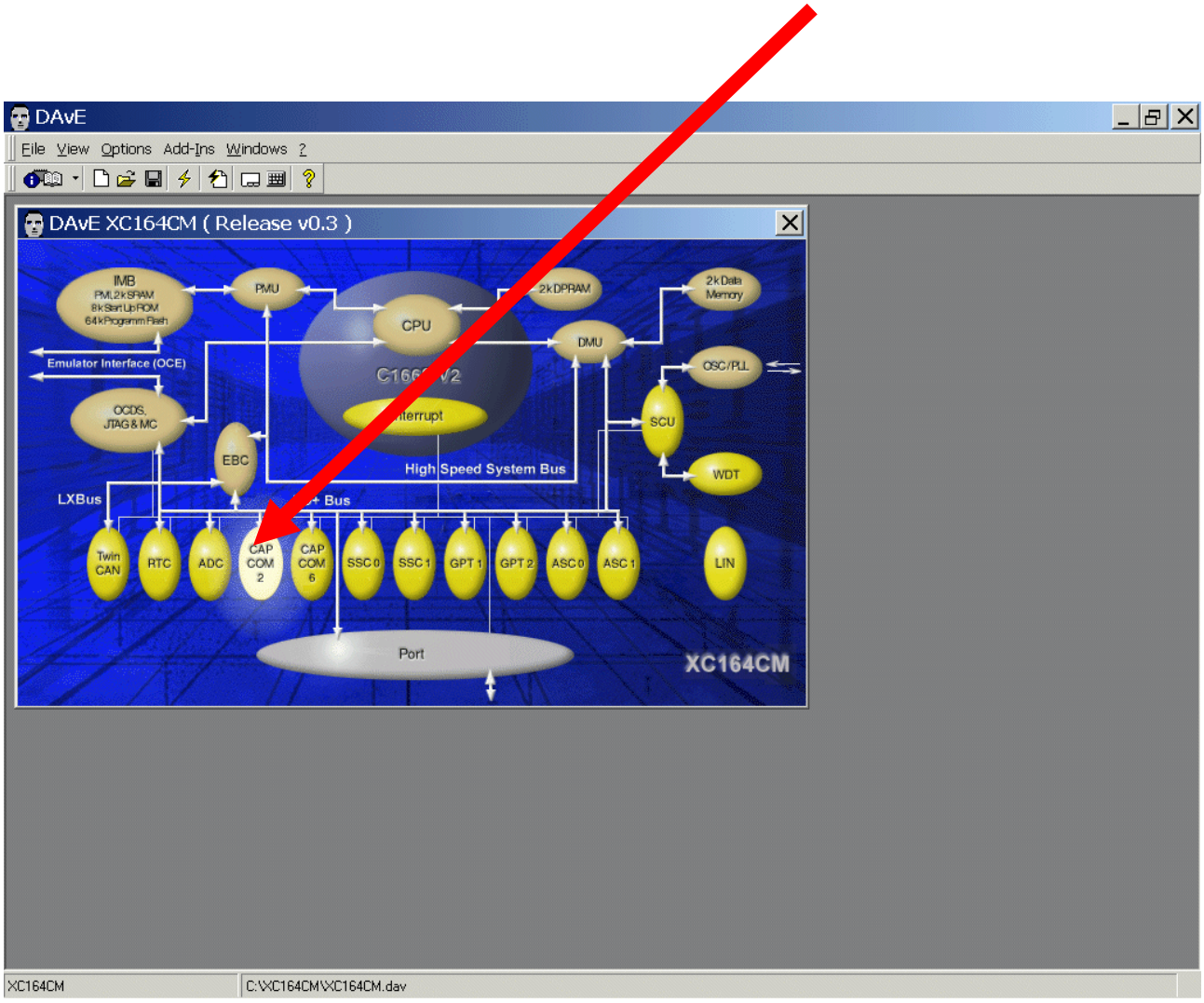
ADC: Functions: Initialization Function: click ✓ ADC_vInit



Exit this dialog now by clicking  the close button.

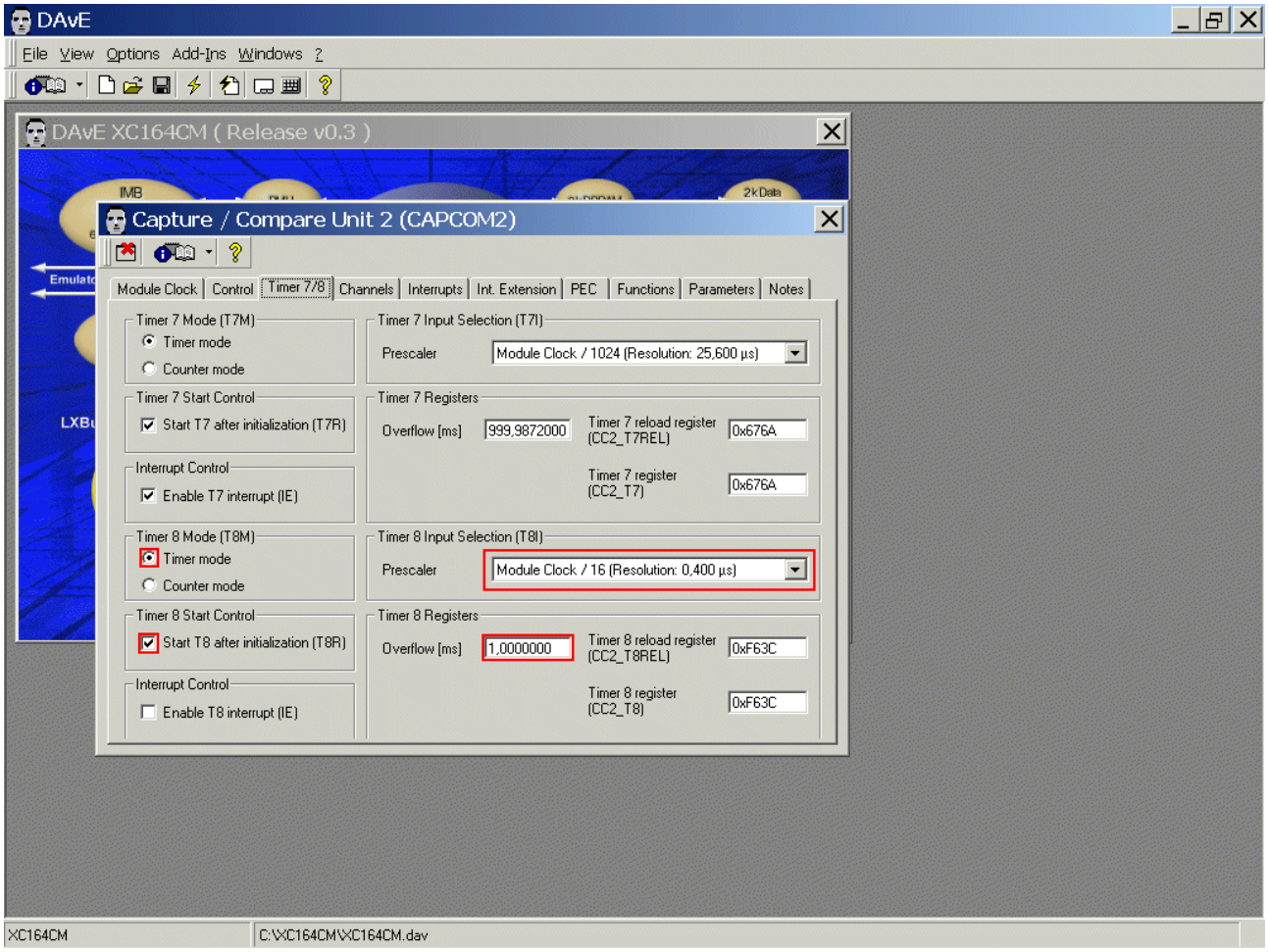
Configuration of the CAPCOM2 (configure as you can see in the screenshots):

The configuration window can be opened by clicking the specific block/module.

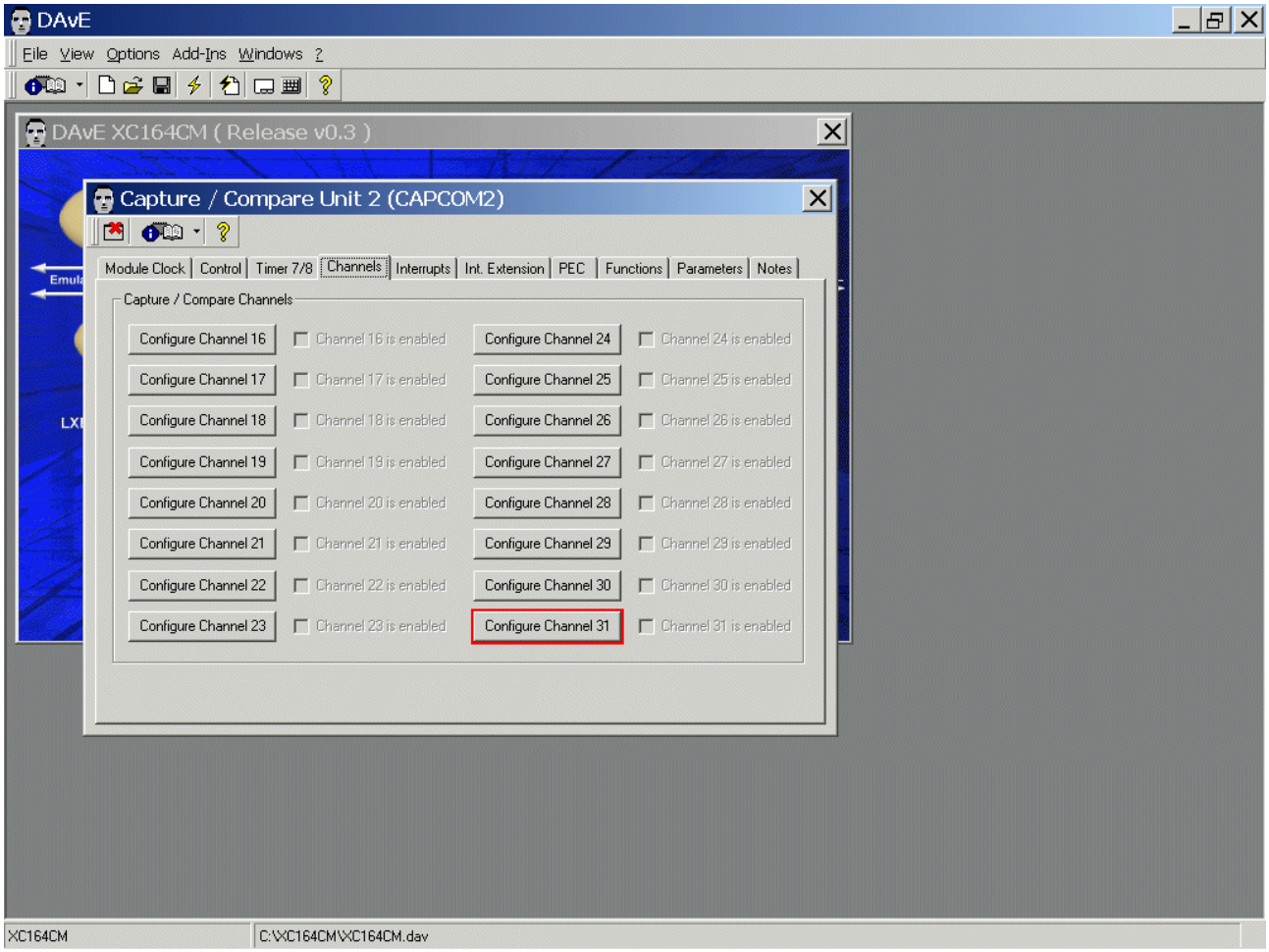


Configure Timer T8 in CAPCOM 2:

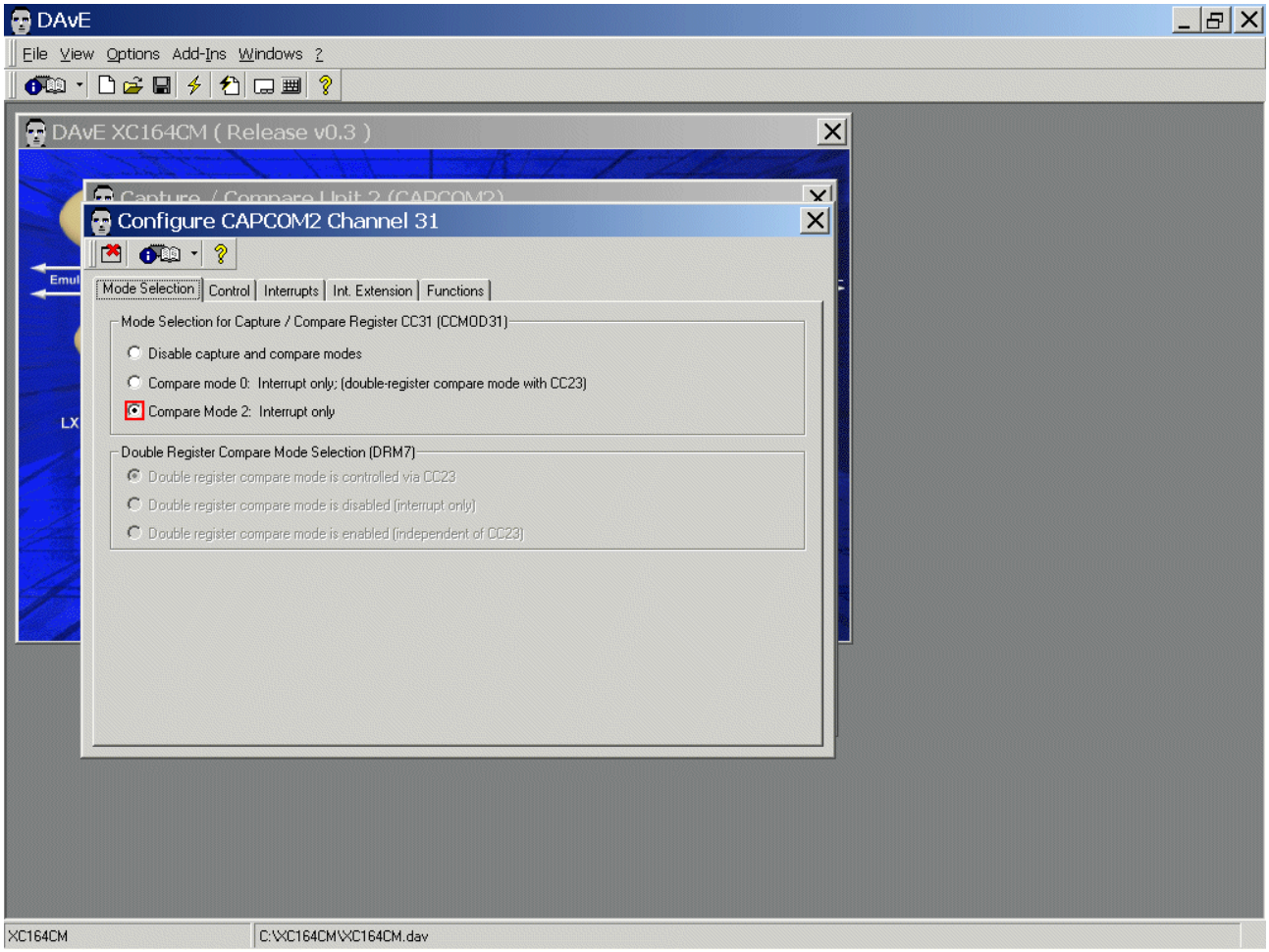
CAPCOM 2: Timer 7/8: Timer 8 Mode: click/check ✓ Timer mode
CAPCOM 2: Timer 7/8: Timer 8 Start Control: click ✓ Start T8 after initialization
CAPCOM 2: Timer 7/8: Timer 8 Input Selection (T7I): Prescaler: choose Module Clock/16
CAPCOM 2: Timer 7/8: Timer 8 Registers: Overflow [ms]: input 1 <ENTER>



CAPCOM 2: Channels: click Configure Channel 31



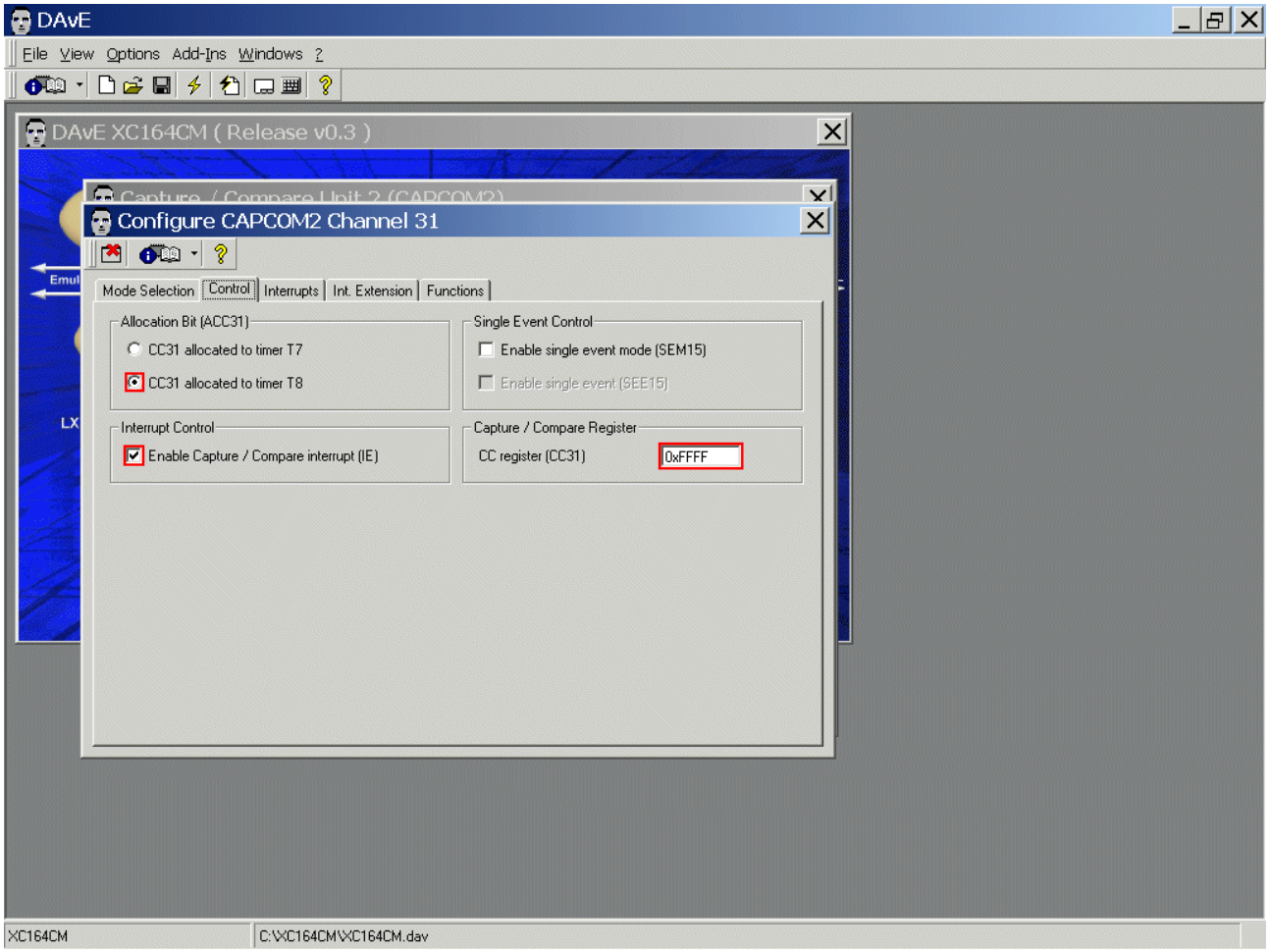
CAPCOM 2: Channels: click Configure Channel 31
Mode Selection: Mode Selection for Capture / Compare Register 31: click ⊙ Compare Mode 2 : Interrupt only
[CAPCOM_2_Channel _31 will trigger (every 1 ms = Timer_8-Overflow) the ADC injected conversion of Analog_Channel_3]



CAPCOM 2: Channels: click Configure Channel 31
Control: Allocation Bit: click ⊙ CC31 allocated to timer T8

CAPCOM 2: Channels: click Configure Channel 31
Control: Interrupt Control: click ✓ Enable Capture / Compare interrupt

CAPCOM 2: Channels: click Configure Channel 31
Control: Capture / Compare Register: CC register (CC31): insert 0xFFFF <ENTER>

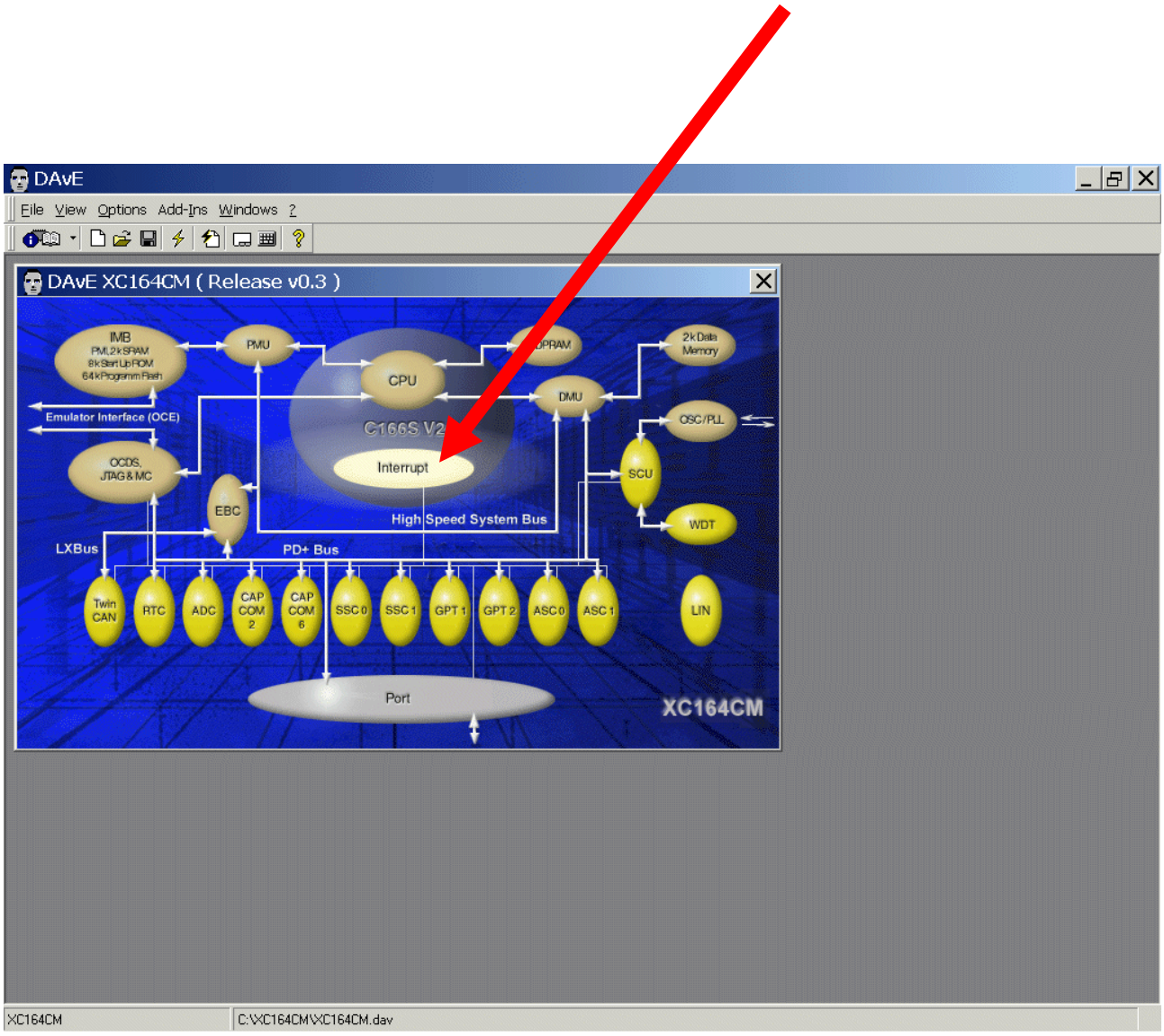


Exit this dialog now by clicking the close button.

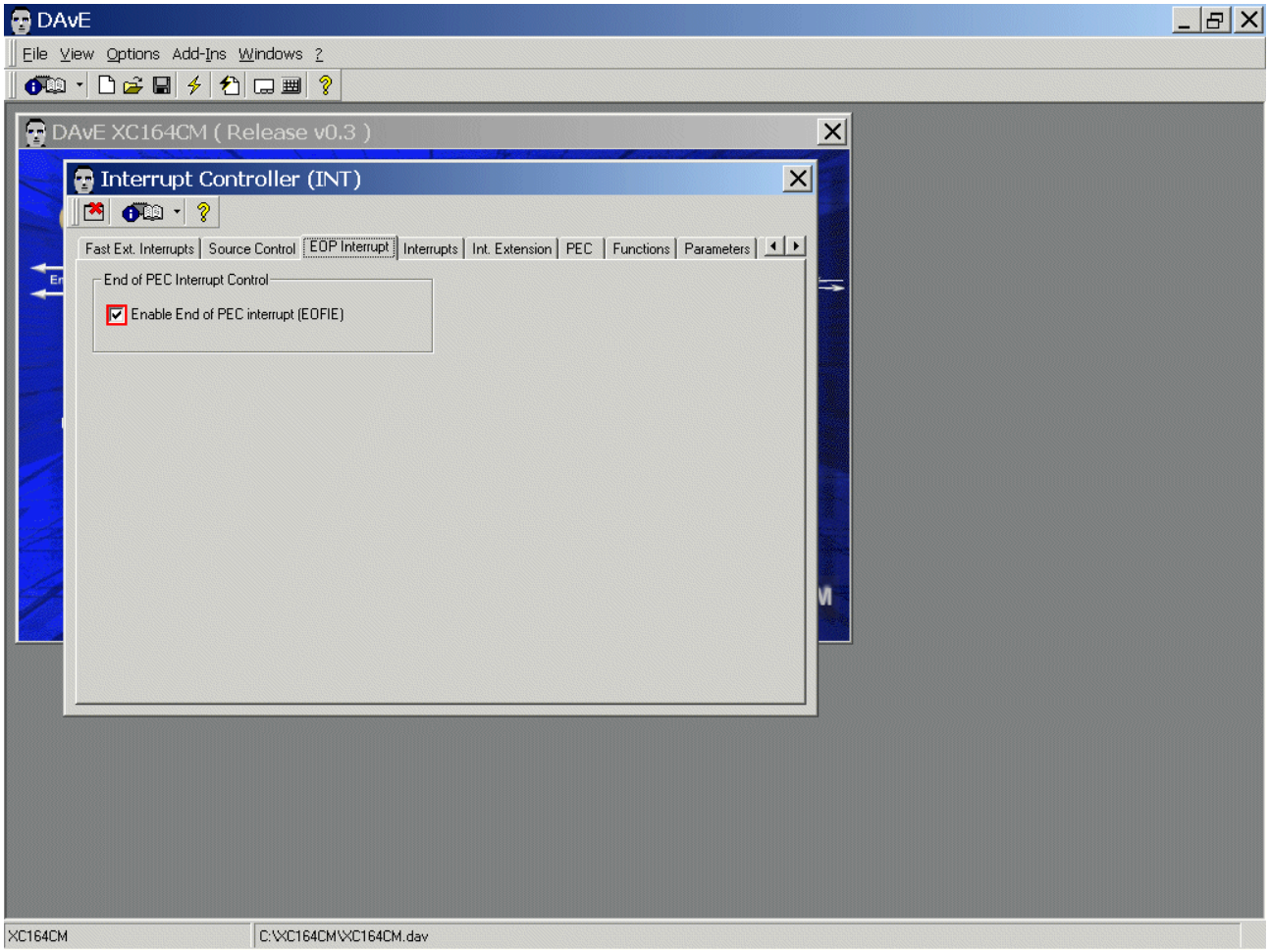
Exit this dialog now by clicking the close button.

Configuration of the Interrupt-System (configure as you can see in the screenshots):

The configuration window can be opened by clicking the specific block/module.

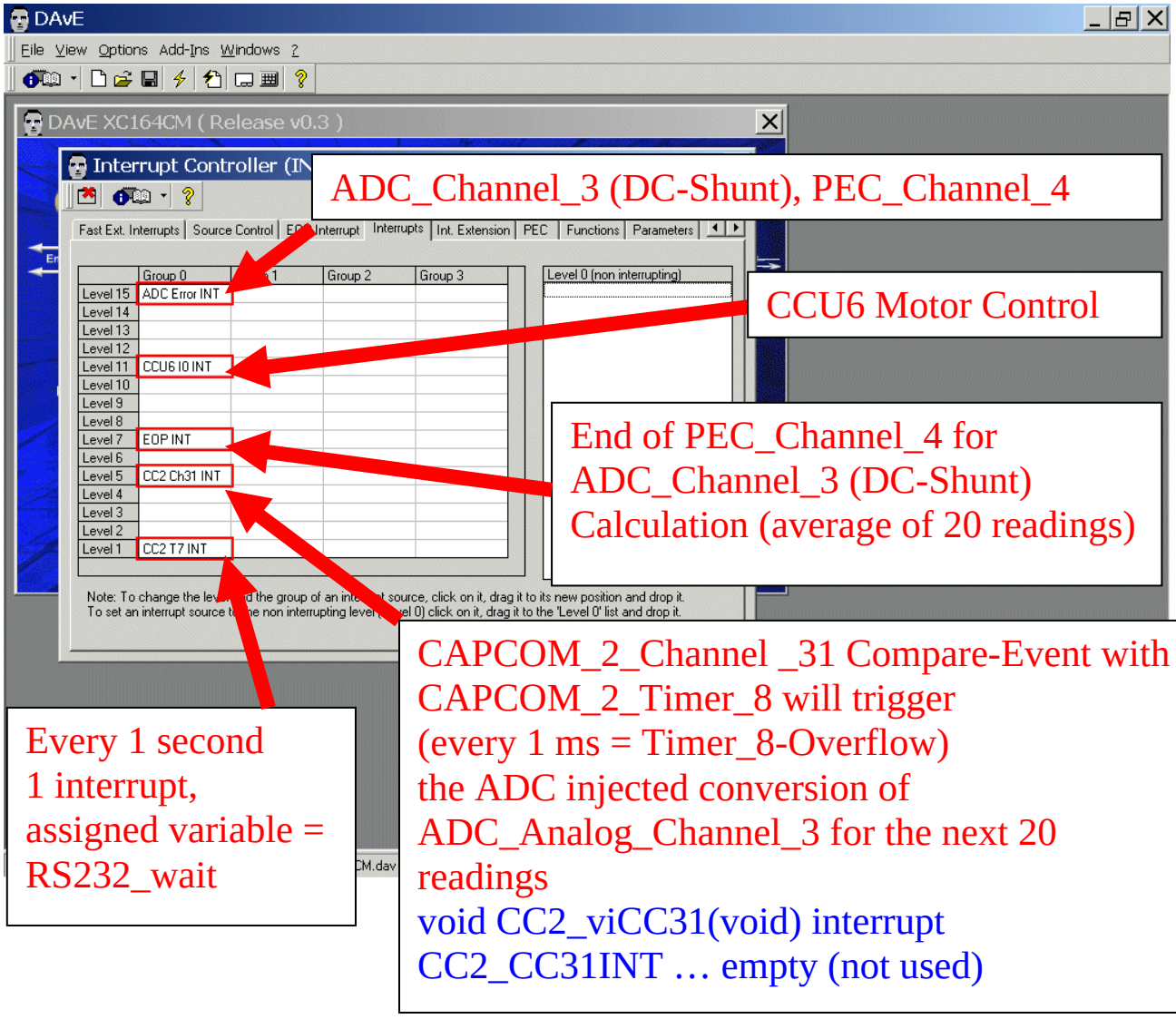


INT: EOP Interrupt: End of PEC Interrupt Control: click ✓ Enable End of PEC interrupt

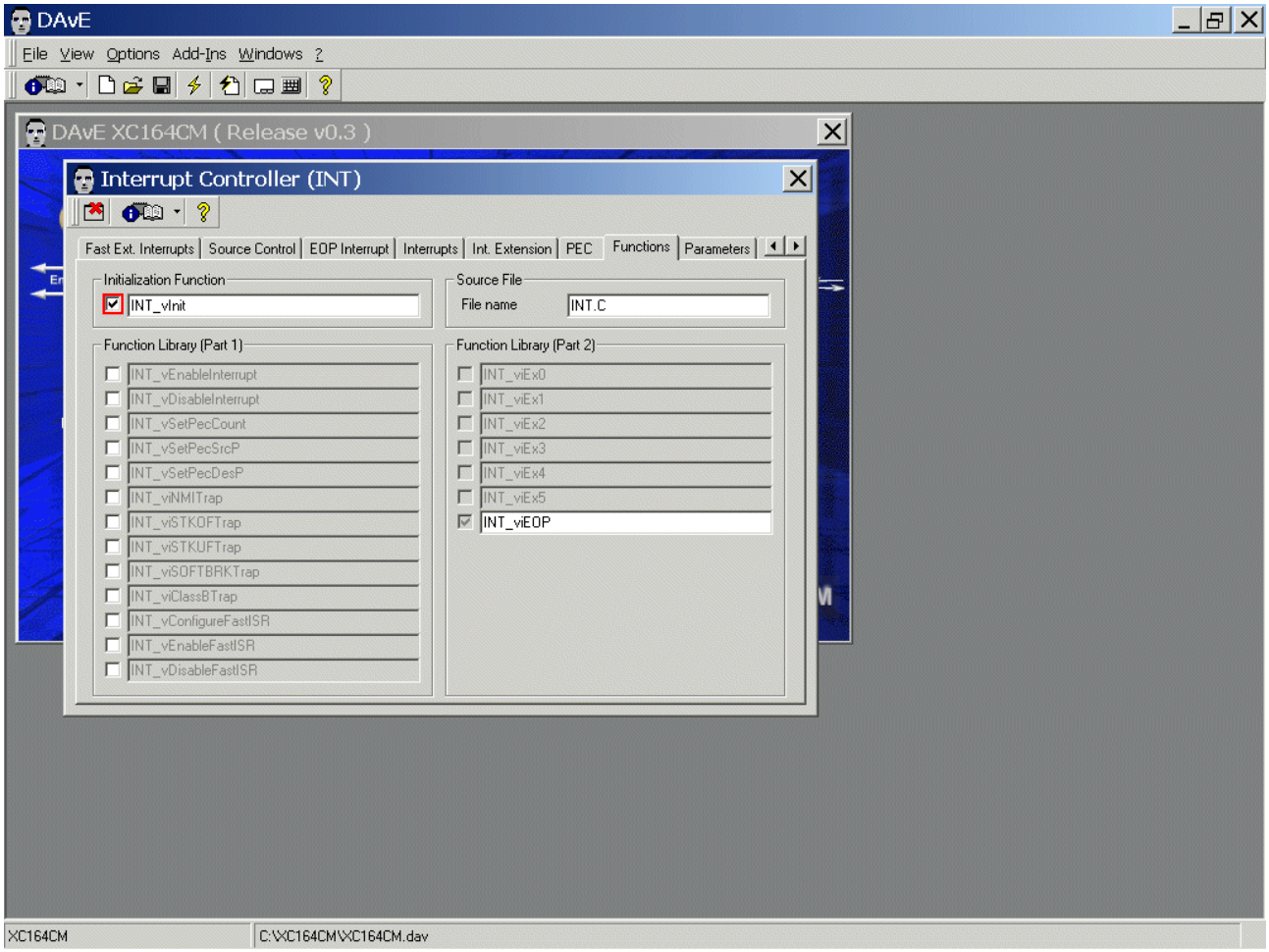


Setup the Interrupt System as shown in the window below:

INT: Interrupts: drag and drop (change priority) CC2 T7 INT to Level 1, Group 0
INT: Interrupts: drag and drop CC2 Ch31 INT to Level 5, Group 0
INT: Interrupts: drag and drop EOP INT to Level 7, Group 0



INT: Functions: Initialization Function: click ✓ INT_vInit

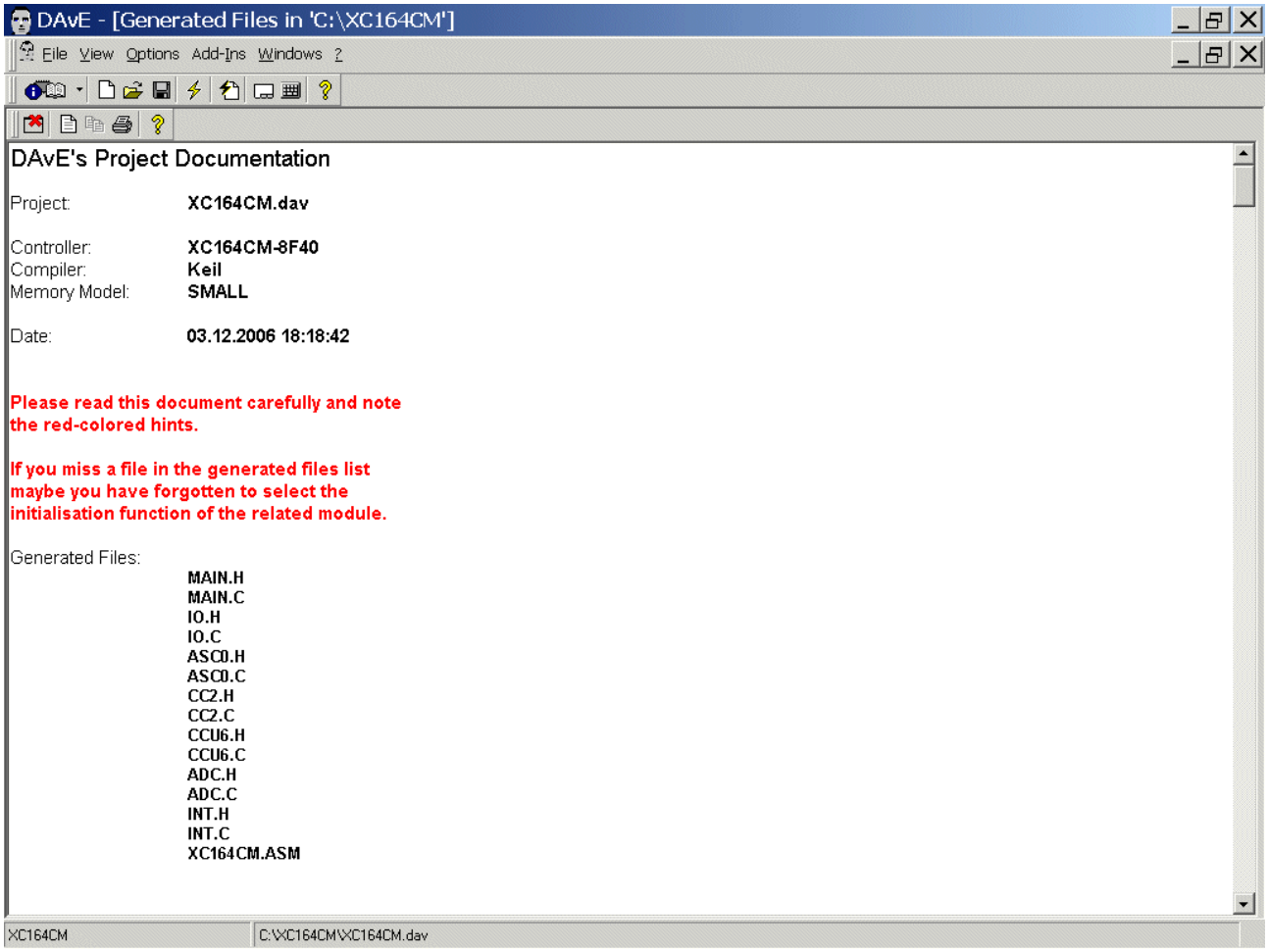


Exit this dialog now by clicking  the close button.

Generate Code:

File Generate Code	or click 

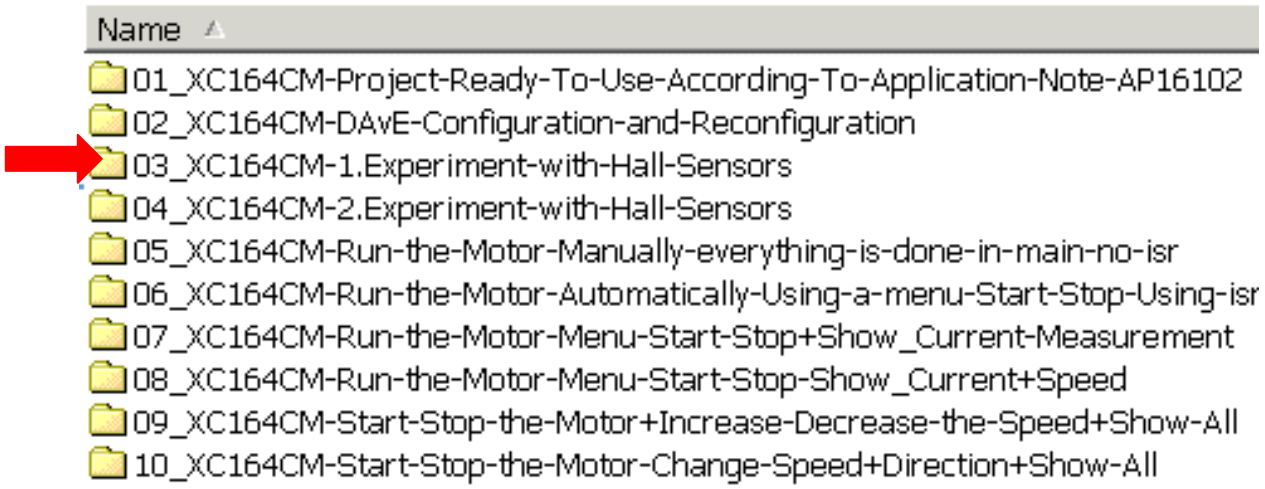
DAvE will show you all the files he has generated
(File Viewer opens automatically).



3.)

1st Experiment with Hall Sensor Signals:

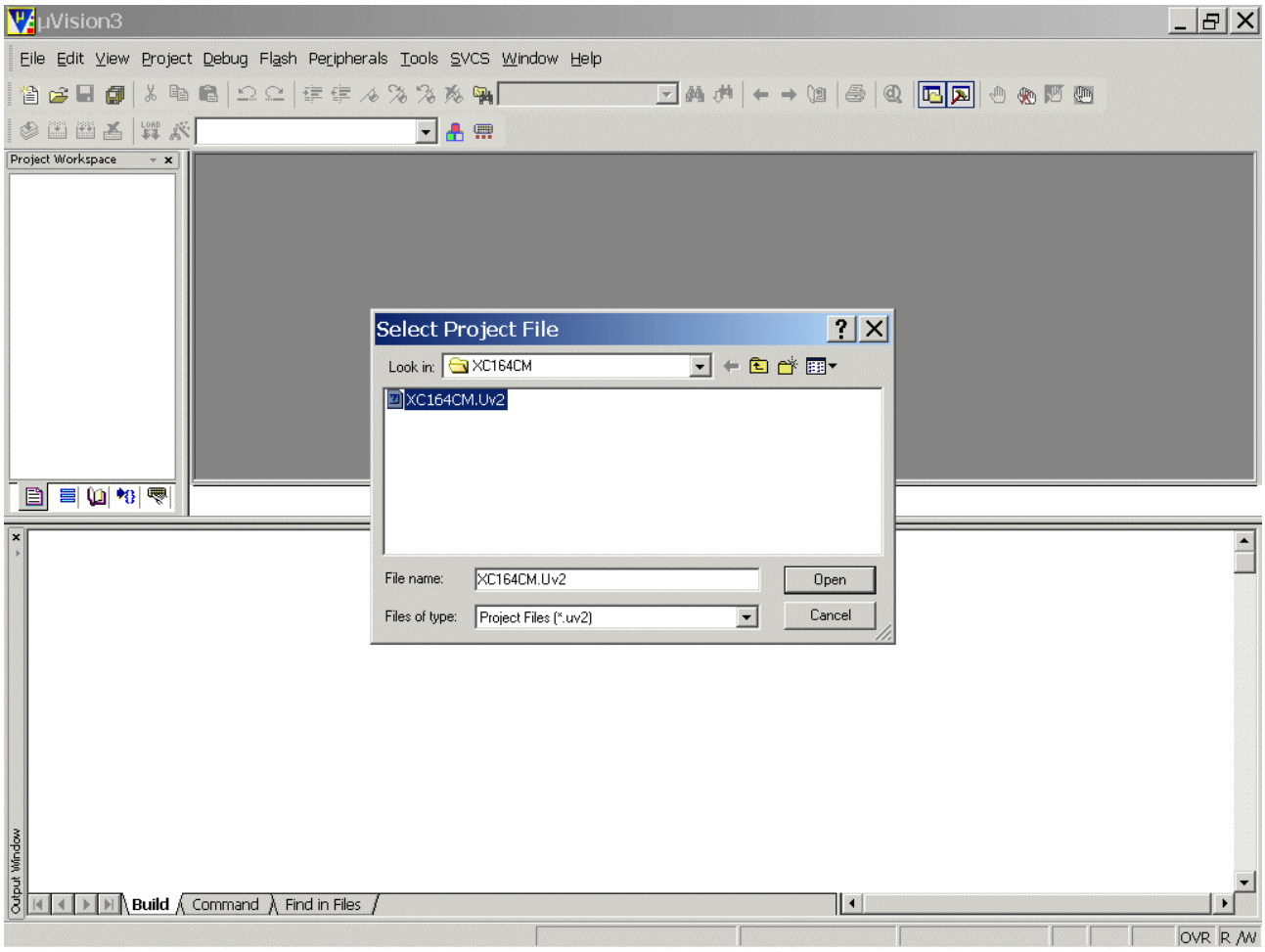
Relevant Project:



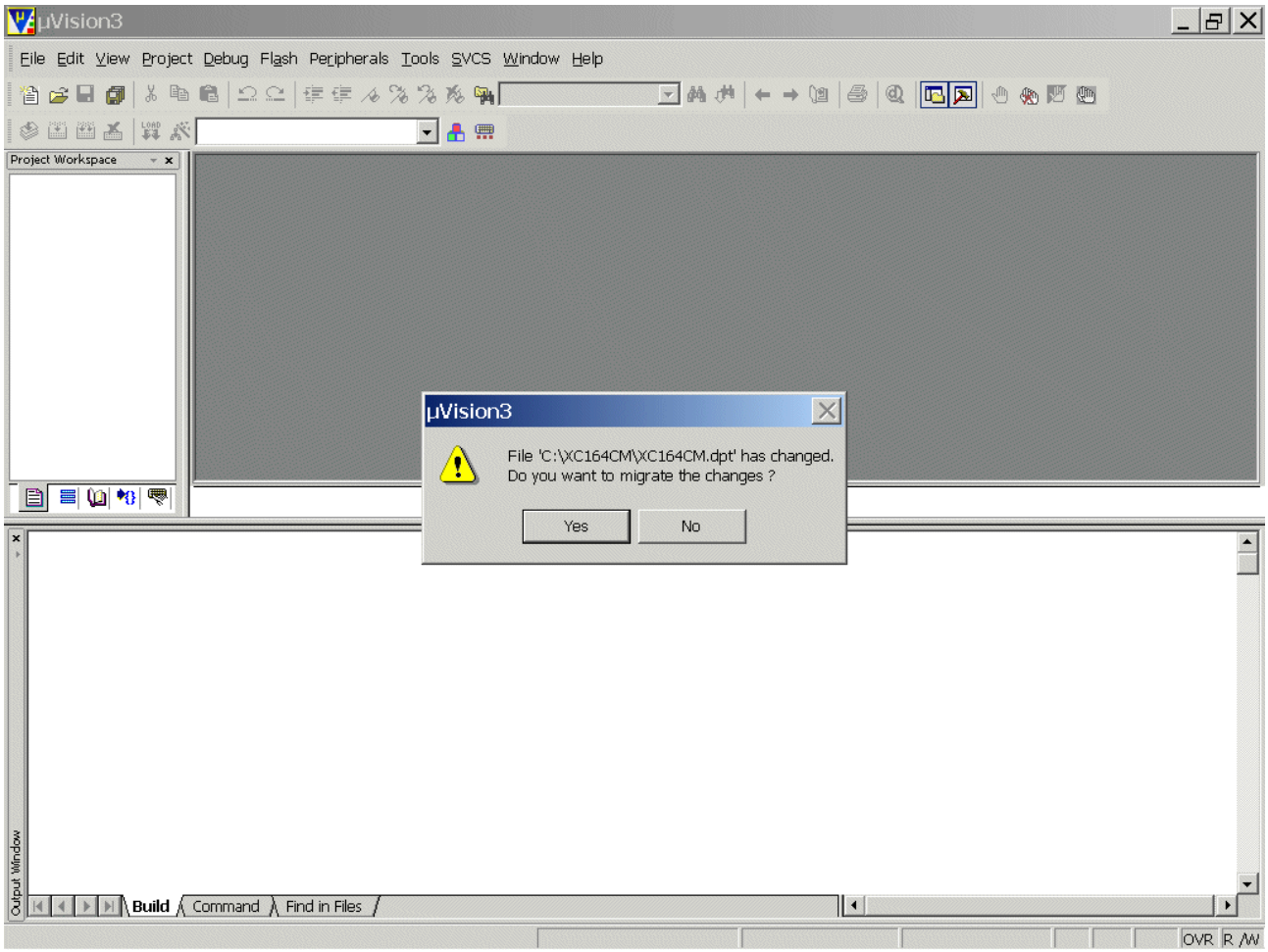
Start Keil µVision3 and open the Project:

If you see an open project – close it: Project - Close Project

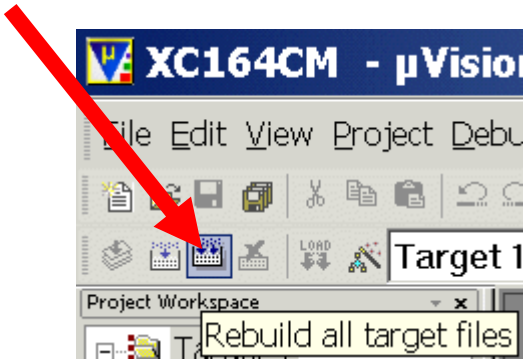
Project - Open Project
Look in: C:\XC164CM
choose: Files of type: Project Files
File name: XC164CM.Uv2

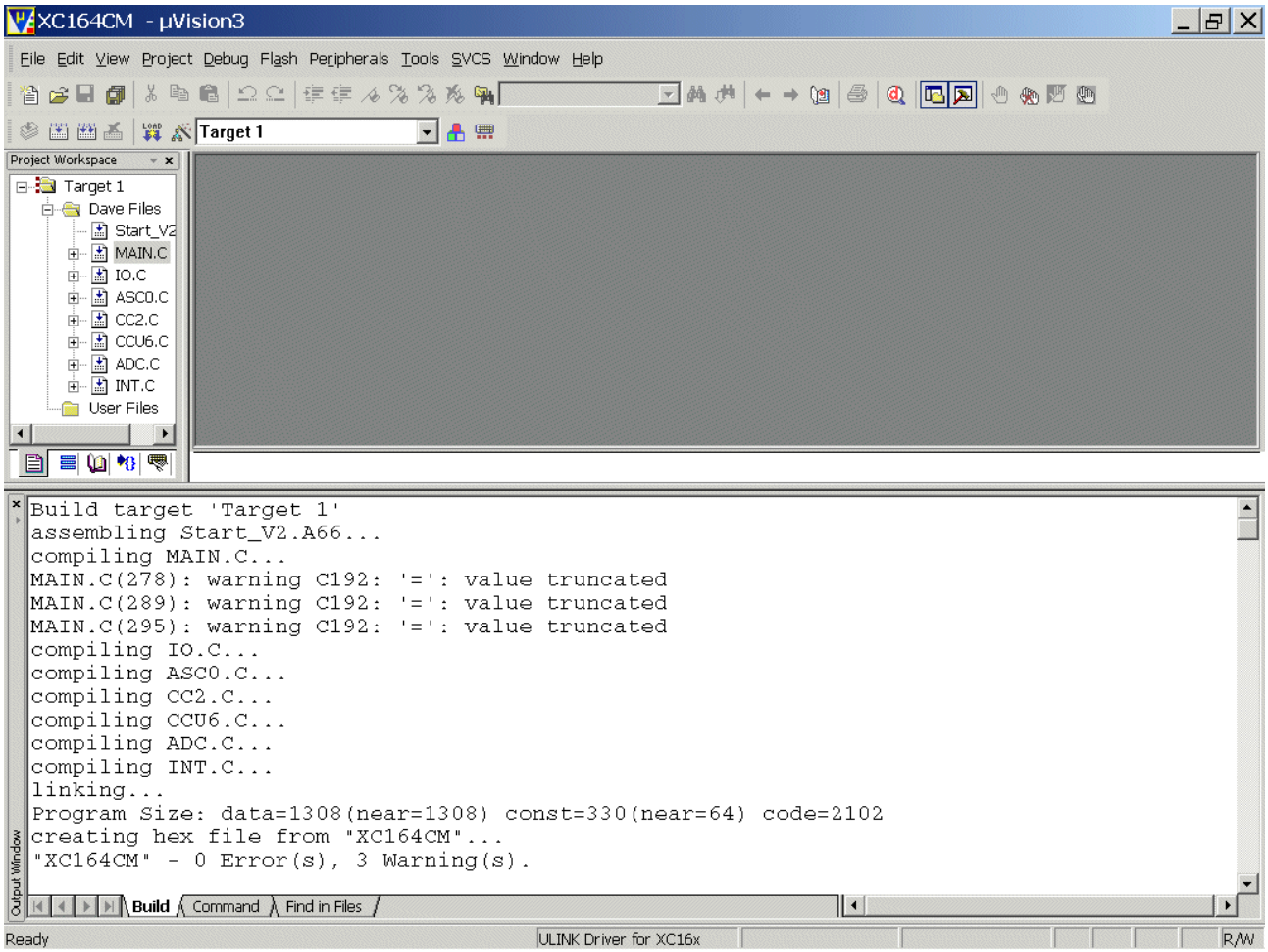


Open

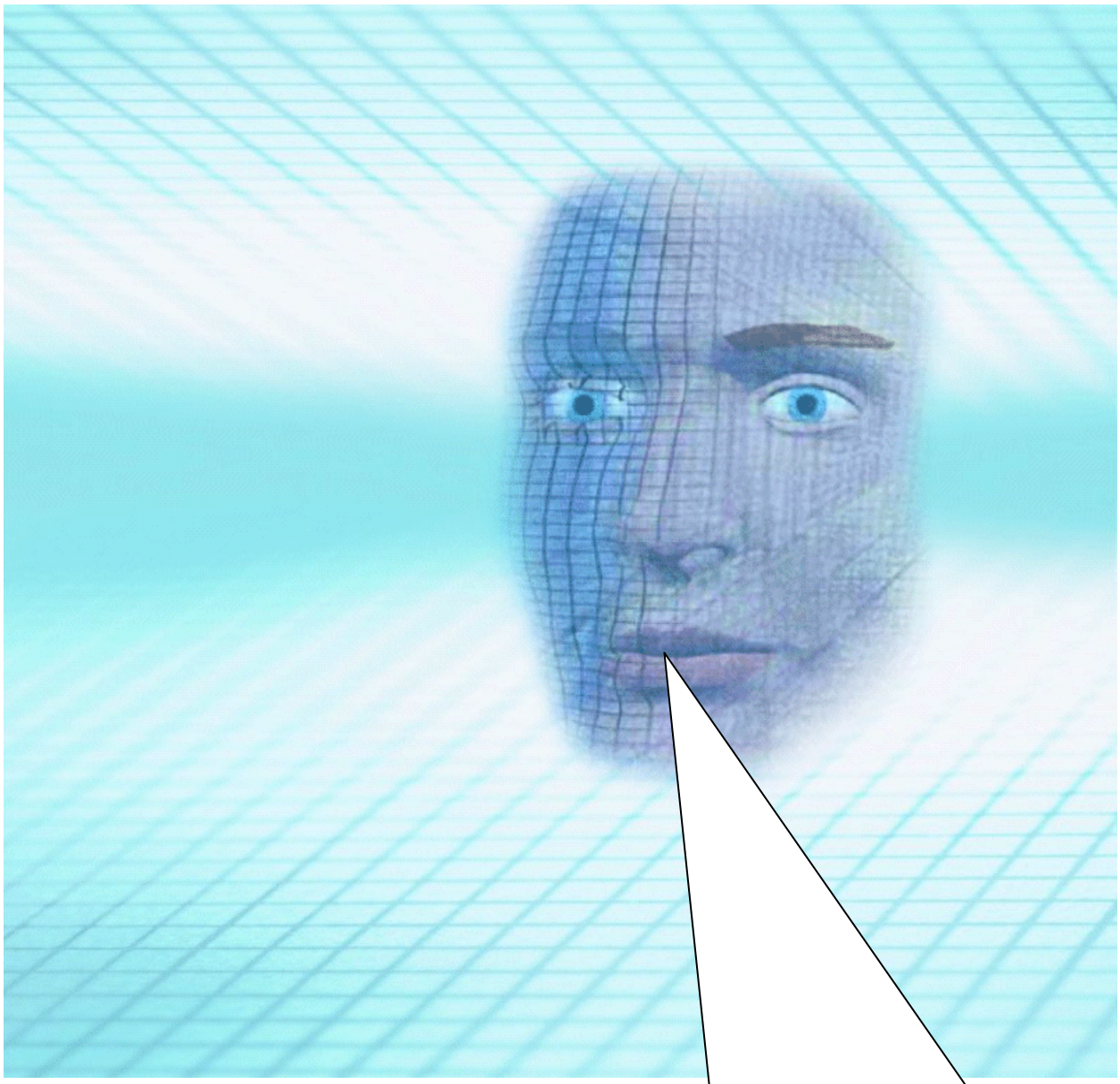

µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
µVision3
File 'C:\XC164CM\XC164CM.dpt' has changed.
Do you want to migrate the changes ?
Yes
No
Output Window
Build
Command
Find in Files
OVR R AW

Click Yes

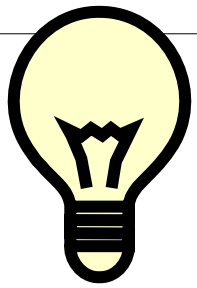
Project – Rebuild all target files	or	click 



Insert your application specific program:

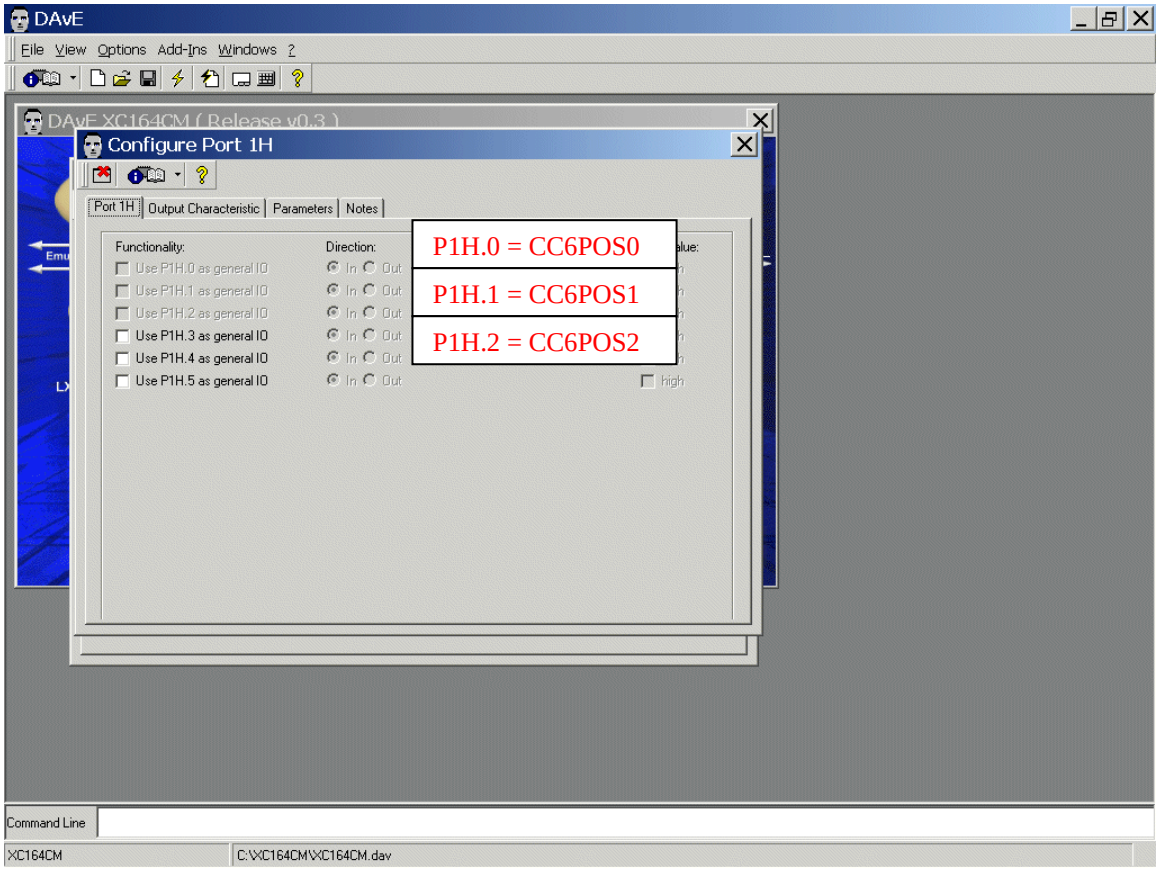


Note:
DAvE doesn't change code which is inserted between '// USER CODE BEGIN' and '// USER CODE END'. Therefore, whenever adding code to DAvE's generated code, write it between '// USER CODE BEGIN' and '// USER CODE END'.
If you wish to change DAvE´s generated code or add code outside these 'USER CODE' sections you will have to insert/modify your changes each time after letting DAvE regenerate code!



Register Information / Additional Information:

Hall Sensor 1 is connected to CC6POS0
Hall Sensor 2 is connected to CC6POS1
Hall Sensor 3 is connected to CC6POS2

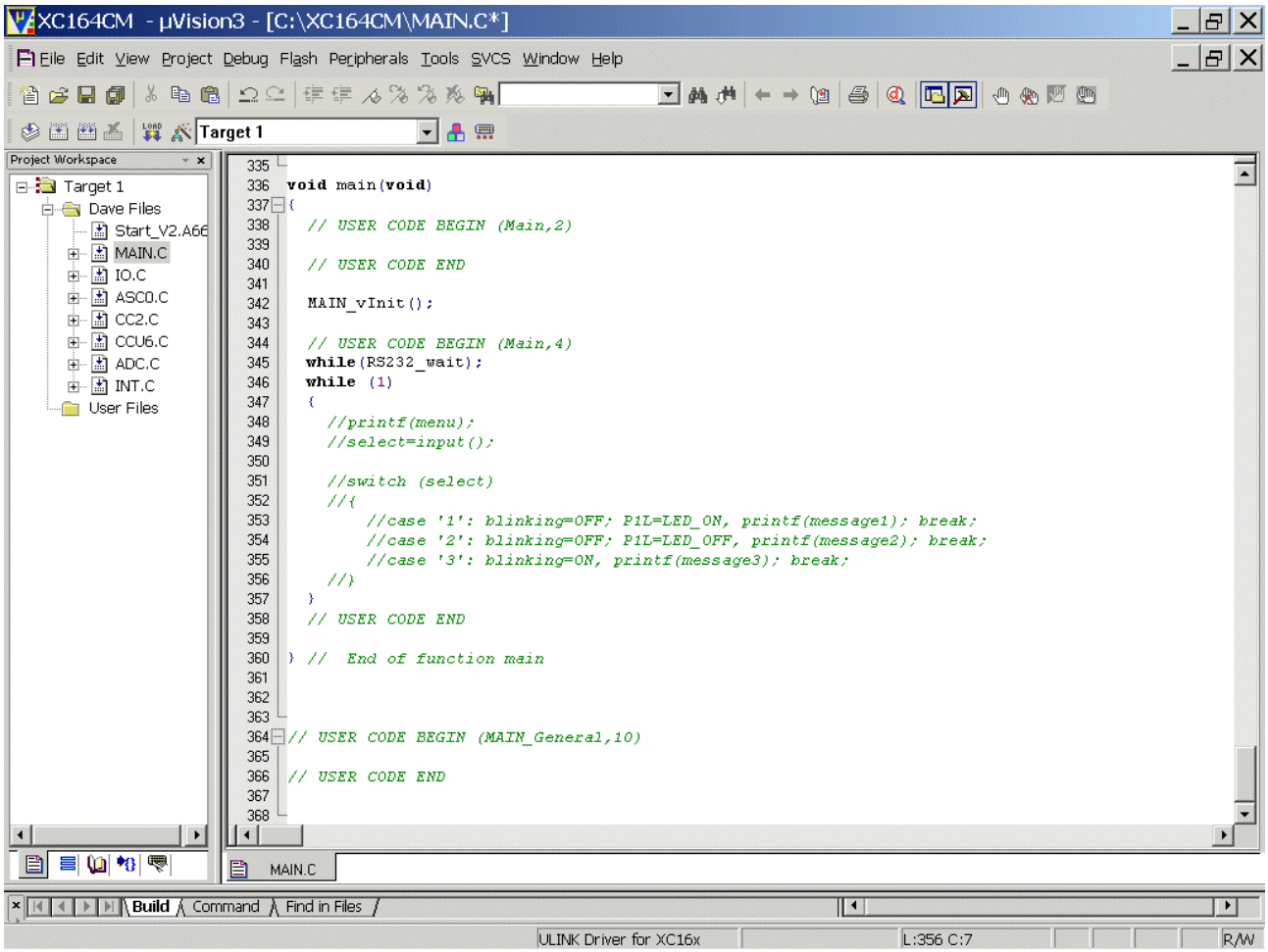


Double click MAIN.C and change Code from:

```
printf(menu);
select=input();

switch (select)
{
   case '1': blinking=OFF; P1L=LED_ON, printf(message1); break;
   case '2': blinking=OFF; P1L=LED_OFF, printf(message2); break;
   case '3': blinking=ON, printf(message3); break;
}
```

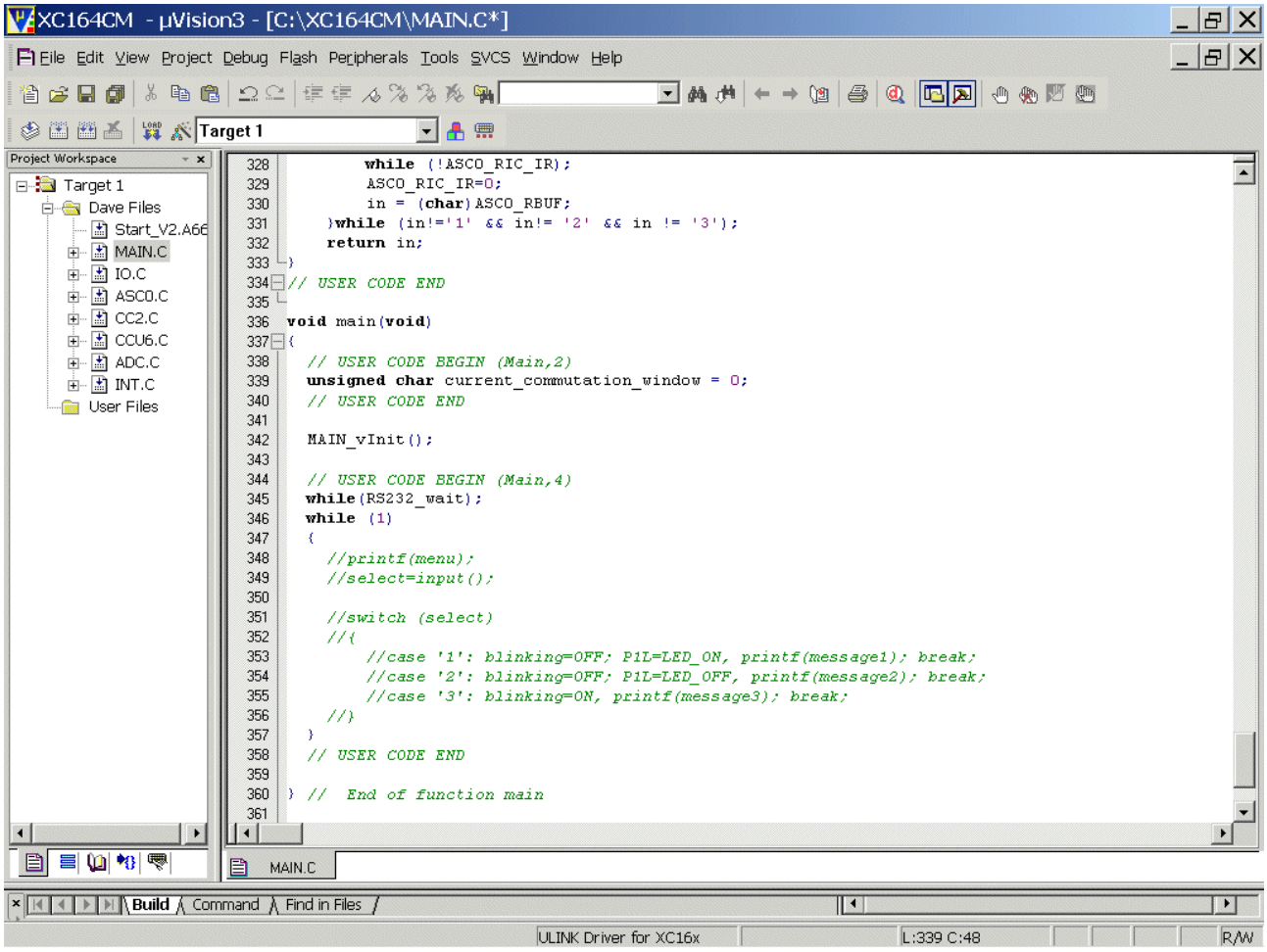
to:

```
//printf(menu);
//select=input();

//switch (select)
//{
   //case '1': blinking=OFF; P1L=LED_ON, printf(message1); break;
   //case '2': blinking=OFF; P1L=LED_OFF, printf(message2); break;
   //case '3': blinking=ON, printf(message3); break;
//}
```


XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A66
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
void main(void)
{
// USER CODE BEGIN (Main,2)
// USER CODE END
MAIN_vInit();
// USER CODE BEGIN (Main,4)
while(RS232_wait);
while (1)
{
//printf(menu);
//select=input();
//switch (select)
//{
//case '1': blinking=OFF; P1L=LED_ON, printf(message1); break;
//case '2': blinking=OFF; P1L=LED_OFF, printf(message2); break;
//case '3': blinking=ON, printf(message3); break;
//}
}
// USER CODE END
} // End of function main
// USER CODE BEGIN (MAIN_General,10)
// USER CODE END
MAIN.C
Build Command Find in Files
ULINK Driver for XC16x
L:356 C:7
R/W

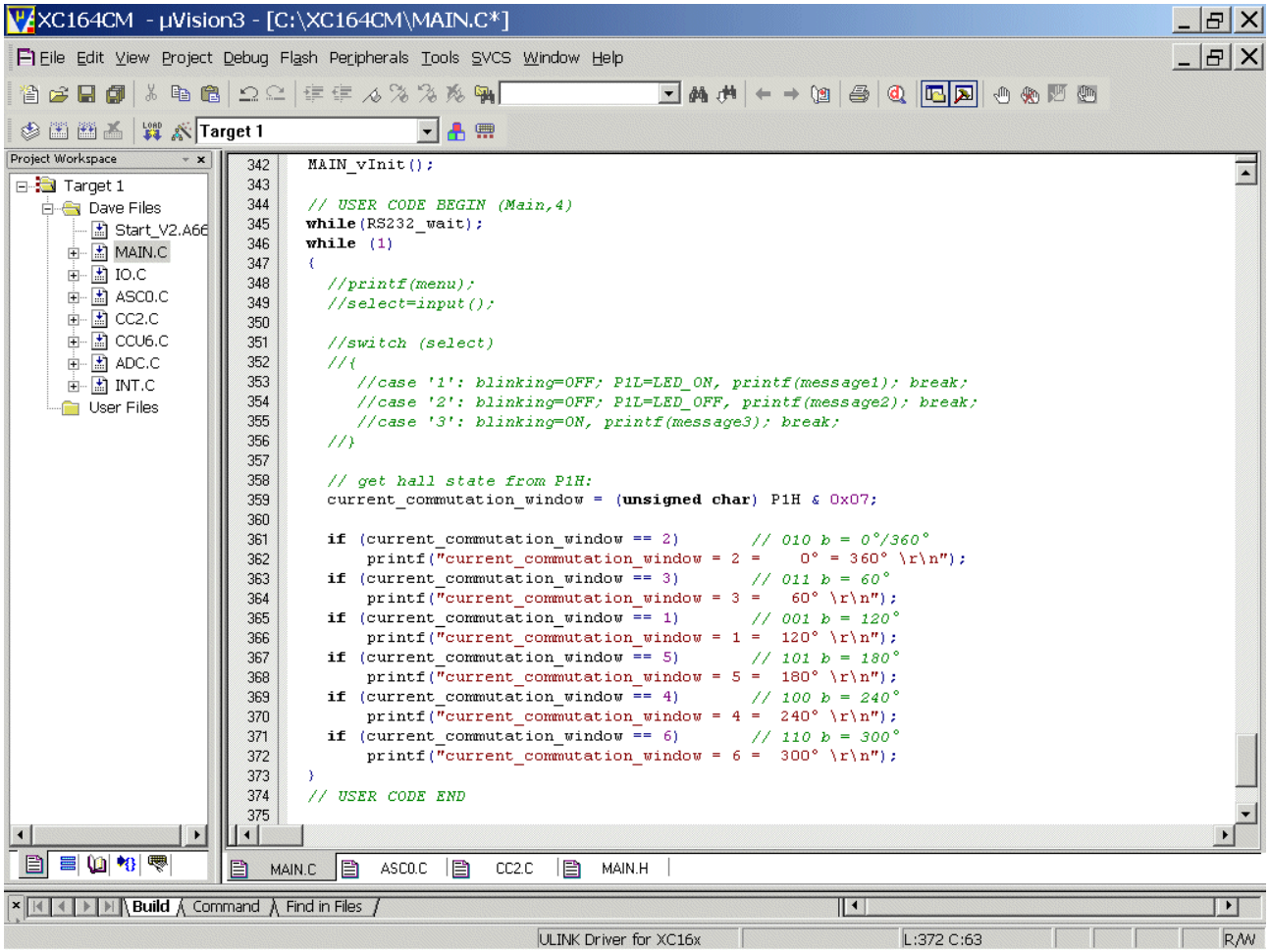
Double click MAIN.C and insert local variable [void main(void) function]:

```
unsigned char current_commutation_window = 0;
```

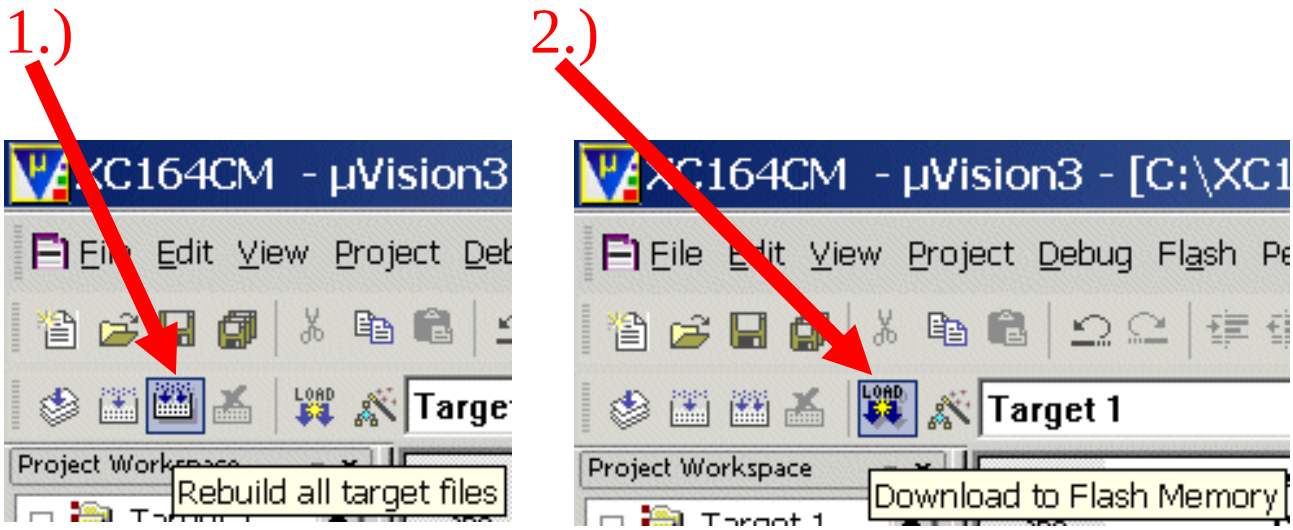


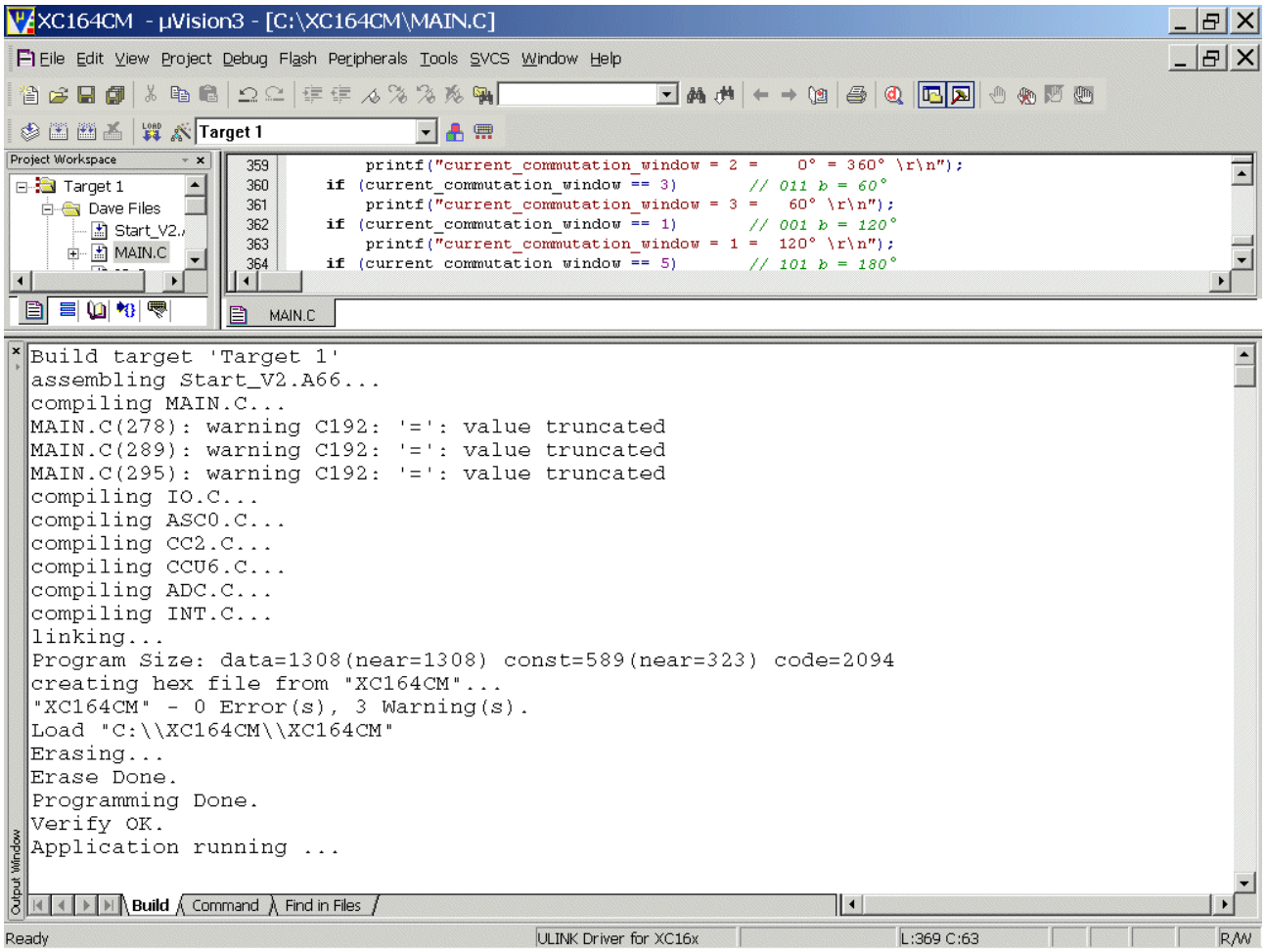
Double click MAIN.C and insert Code [void main(void) function]:

```
    // get hall state from P1H:
    current_commutation_window = (unsigned char) P1H & 0x07;

    if (current_commutation_window == 2)       // 010 b = 0°/360°
        printf("current_commutation_window = 2 =    0° = 360° \r\n");
    if (current_commutation_window == 3)       // 011 b = 60°
        printf("current_commutation_window = 3 =   60° \r\n");
    if (current_commutation_window == 1)       // 001 b = 120°
        printf("current_commutation_window = 1 =  120° \r\n");
    if (current_commutation_window == 5)       // 101 b = 180°
        printf("current_commutation_window = 5 =  180° \r\n");
    if (current_commutation_window == 4)       // 100 b = 240°
        printf("current_commutation_window = 4 =  240° \r\n");
    if (current_commutation_window == 6)       // 110 b = 300°
        printf("current_commutation_window = 6 =  300° \r\n");
```



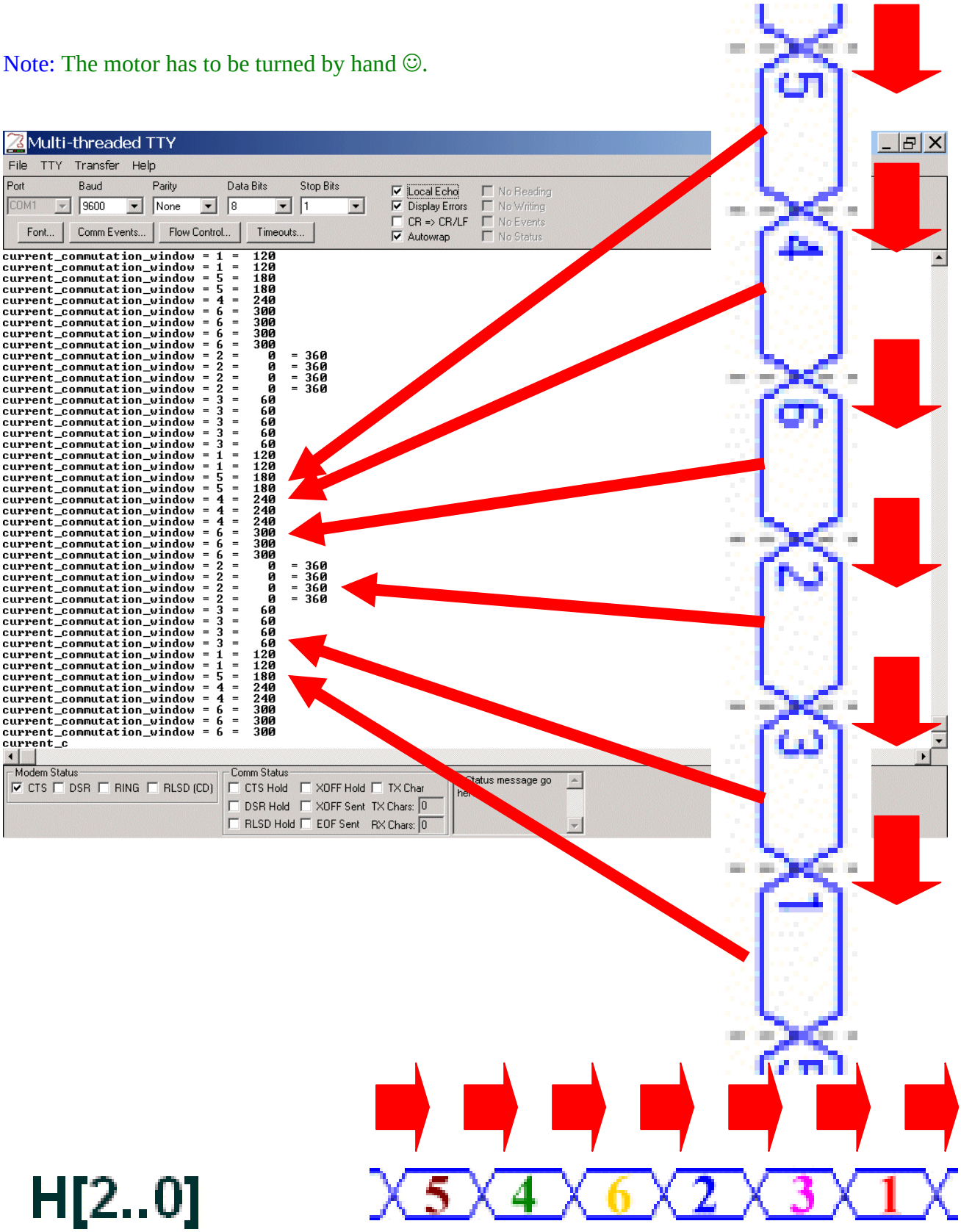
Build Application:





And see the result:

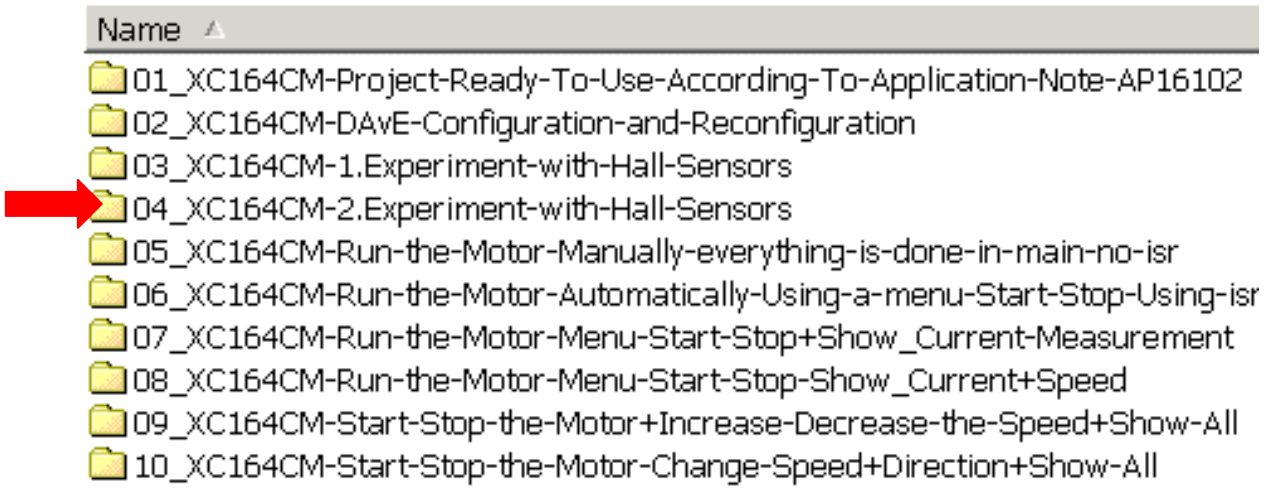
Note: The motor has to be turned by hand ☺.



4.)

2nd Experiment with Hall Sensor Signals:

Relevant Project:

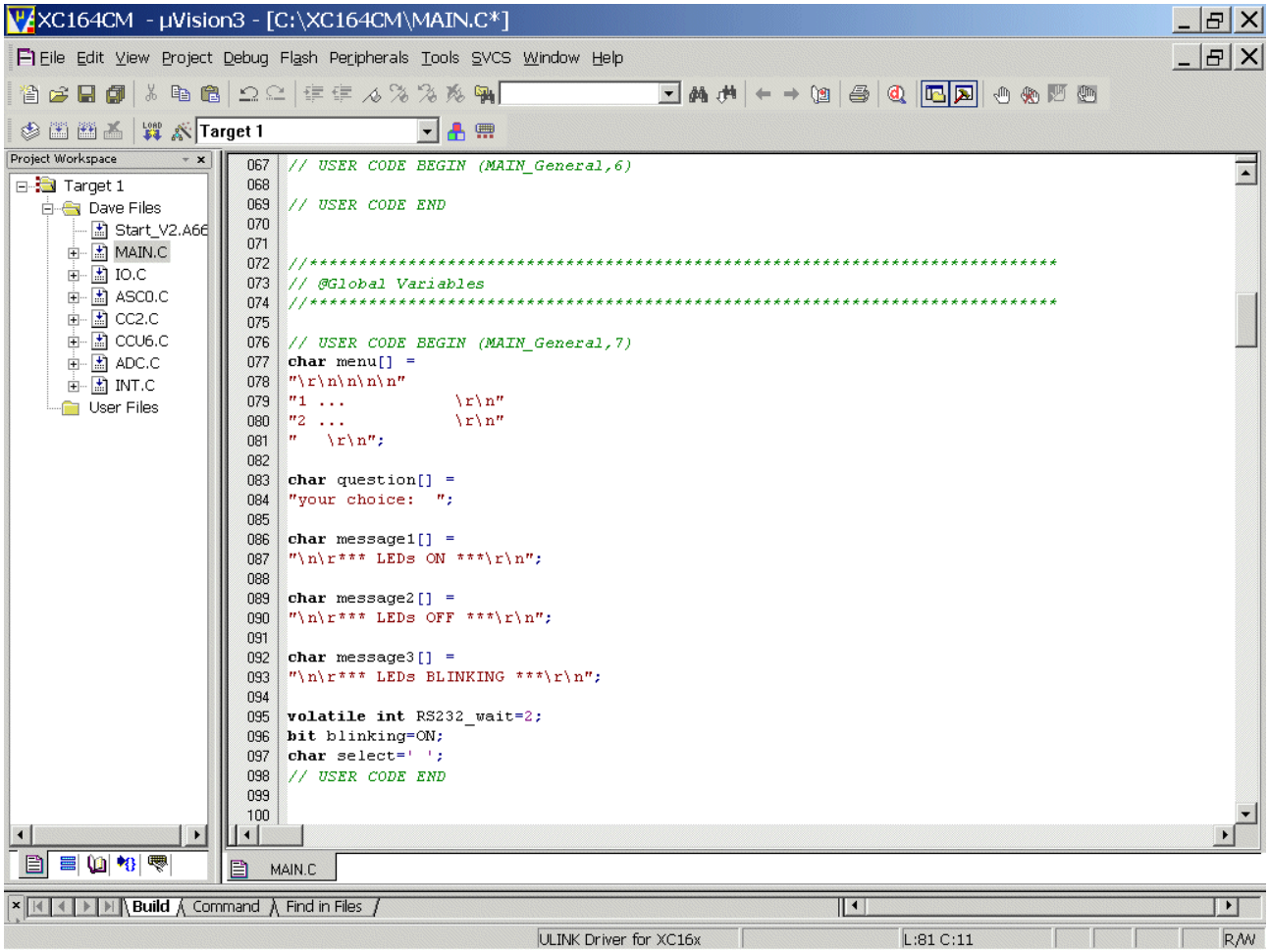


Double click MAIN.C and change Global Variable from:

```
char menu[] =
"\r\n\n\n\n"
"1 ... LEDs P1L.1+P1L.3+P1L.5+P1L.7 ON\r\n"
"2 ... LEDs P1L.1+P1L.3+P1L.5+P1L.7 OFF\r\n"
"3 ... LEDs P1L.1+P1L.3+P1L.5+P1L.7 blinking\r\n"
"   \r\n";
```

to:

```
char menu[] =
"\r\n\n\n\n"
"1 ...          \r\n"
"2 ...          \r\n"
"   \r\n";
```

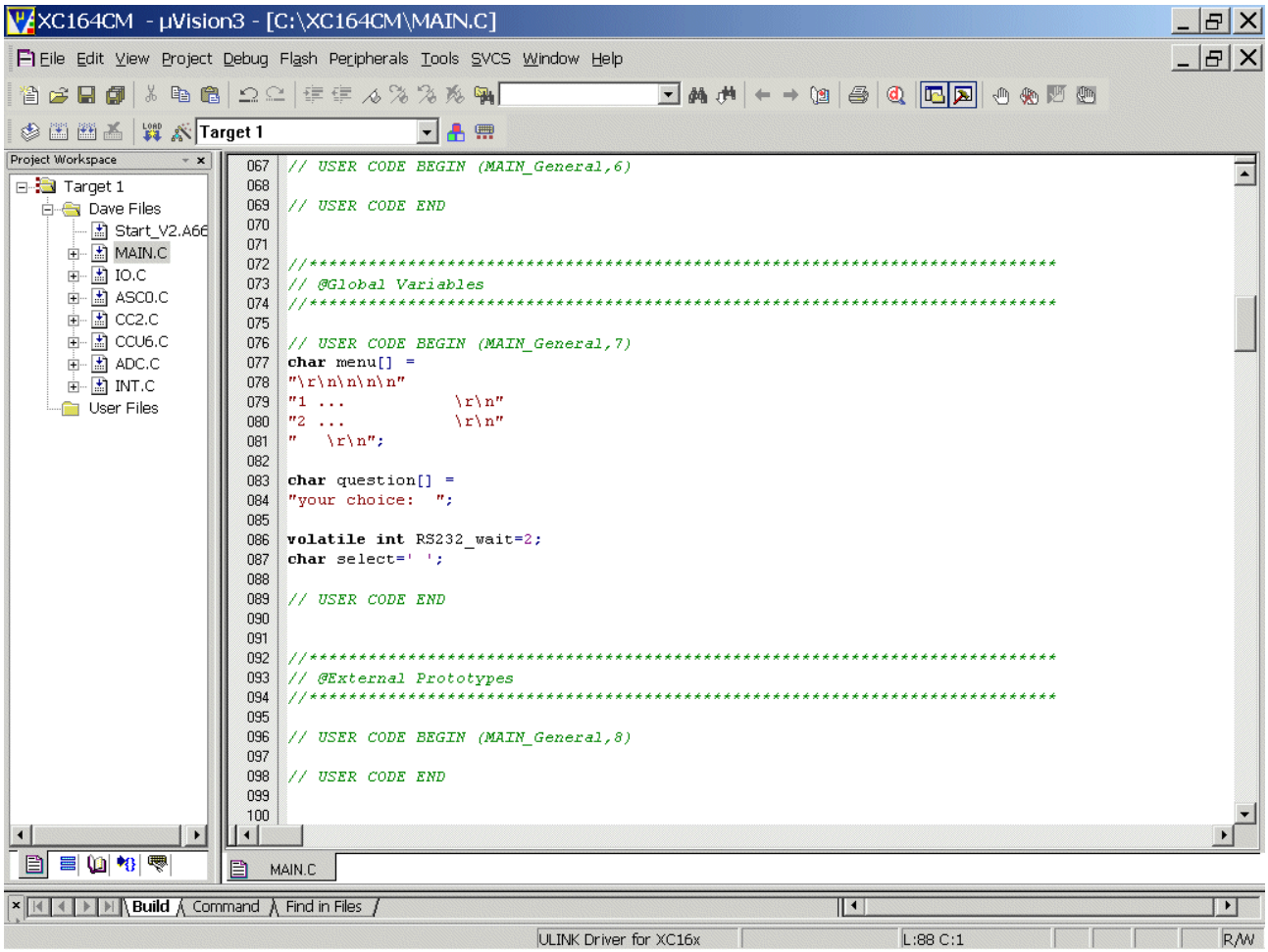


Double click MAIN.C and DELETE Global Variables:

```
char message1[] =
"\n\r*** LEDs ON ***\r\n";

char message2[] =
"\n\r*** LEDs OFF ***\r\n";

char message3[] =
"\n\r*** LEDs BLINKING ***\r\n";
```

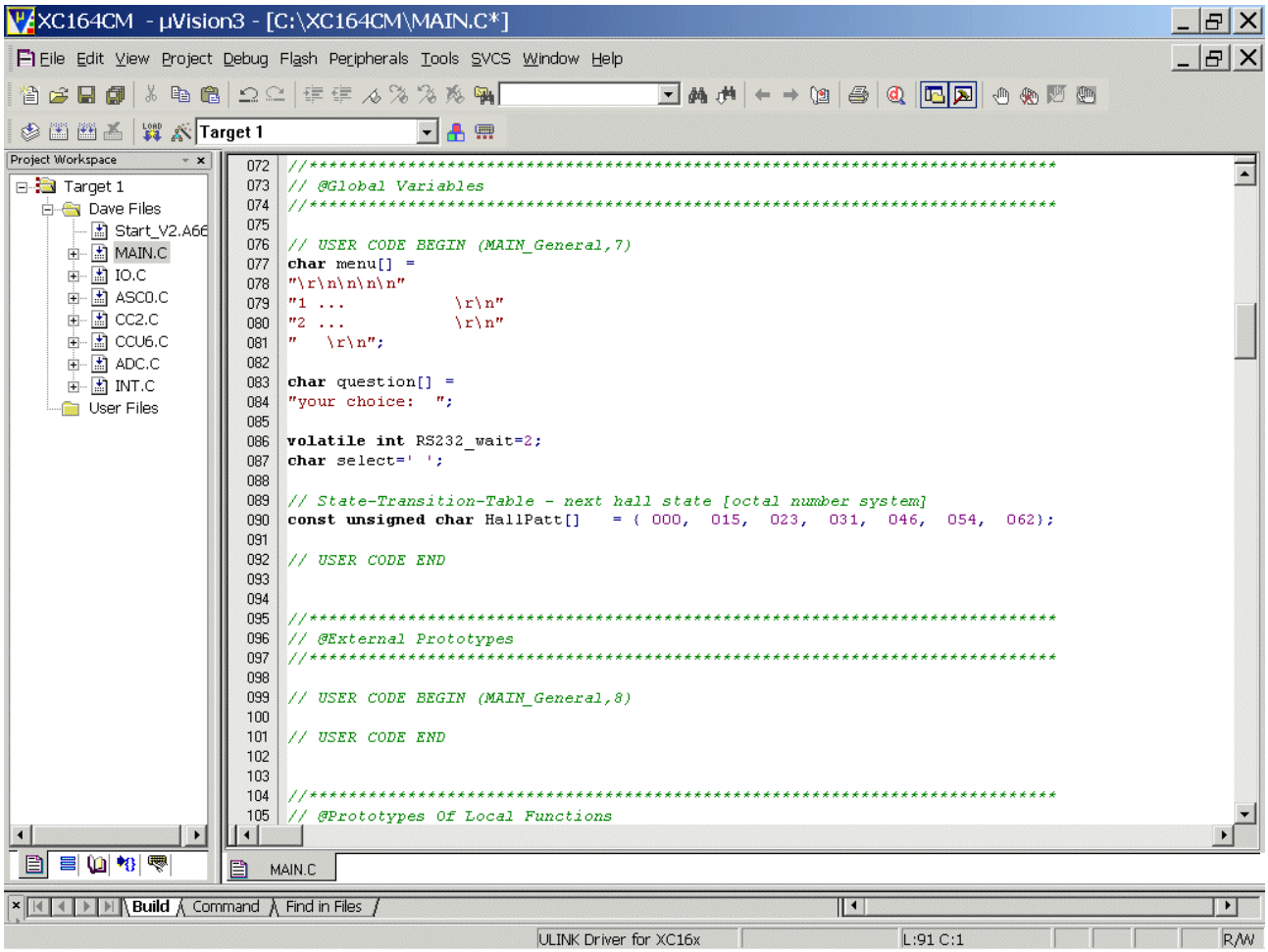
Double click MAIN.C and DELETE Global Variable:

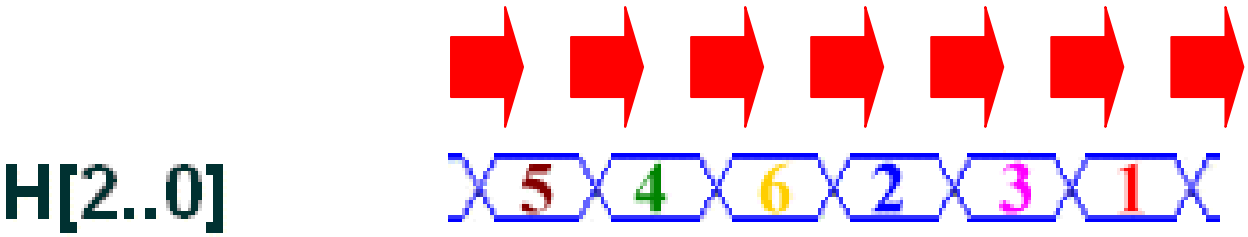
```
bit blinking=ON;
```



Double click MAIN.C and insert Global Variable:

```
// State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt[]   = { 000,  015,  023,  031,  046,  054,  062};
```



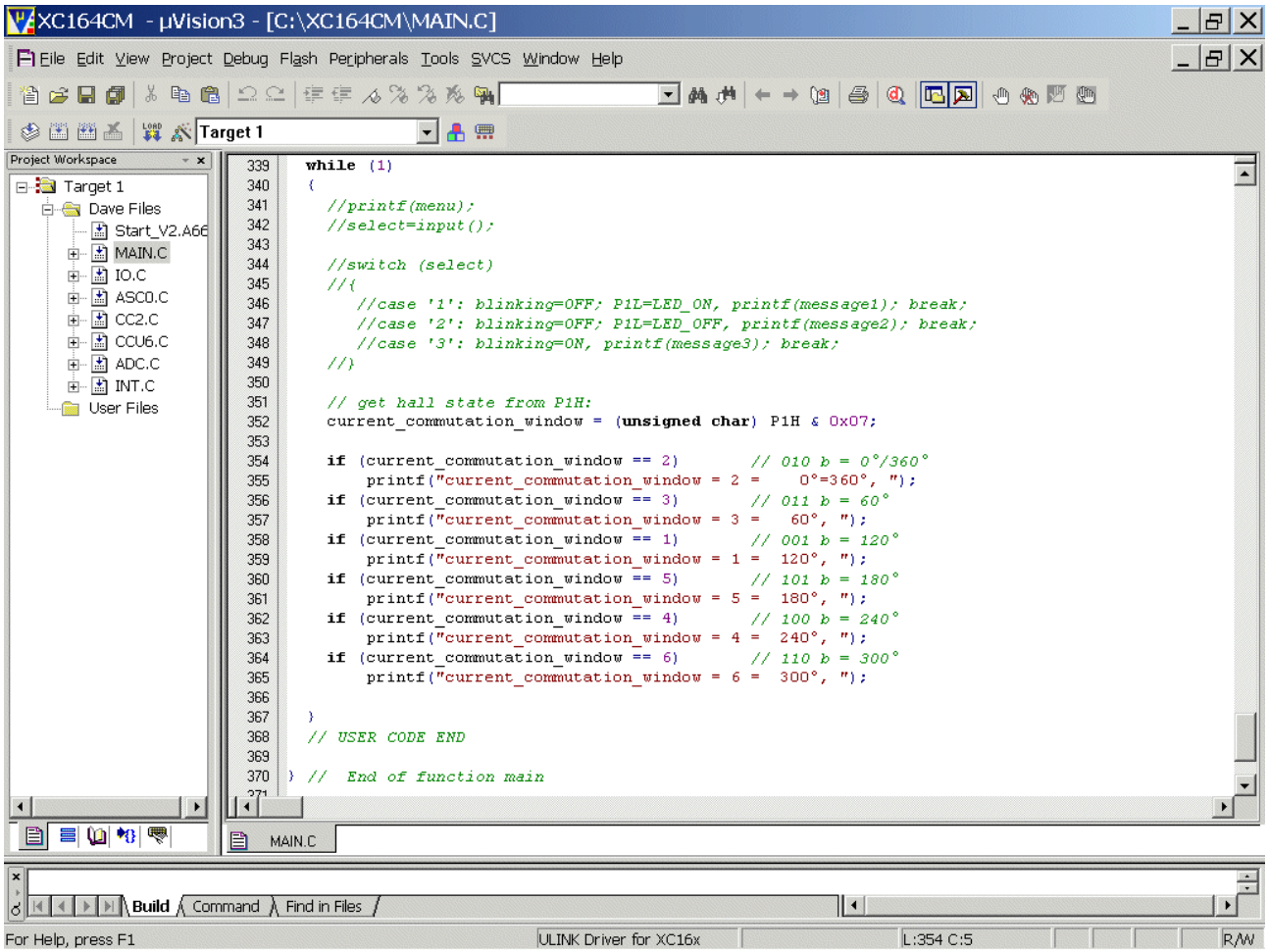


Double click MAIN.C and change Code from:

```
    if (current_commutation_window == 2)       // 010 b = 0°/360°
        printf("current_commutation_window = 2 =    0° = 360° \r\n");
    if (current_commutation_window == 3)       // 011 b = 60°
        printf("current_commutation_window = 3 =   60° \r\n");
    if (current_commutation_window == 1)       // 001 b = 120°
        printf("current_commutation_window = 1 =  120° \r\n");
    if (current_commutation_window == 5)       // 101 b = 180°
        printf("current_commutation_window = 5 =  180° \r\n");
    if (current_commutation_window == 4)       // 100 b = 240°
        printf("current_commutation_window = 4 =  240° \r\n");
    if (current_commutation_window == 6)       // 110 b = 300°
        printf("current_commutation_window = 6 =  300° \r\n");
```

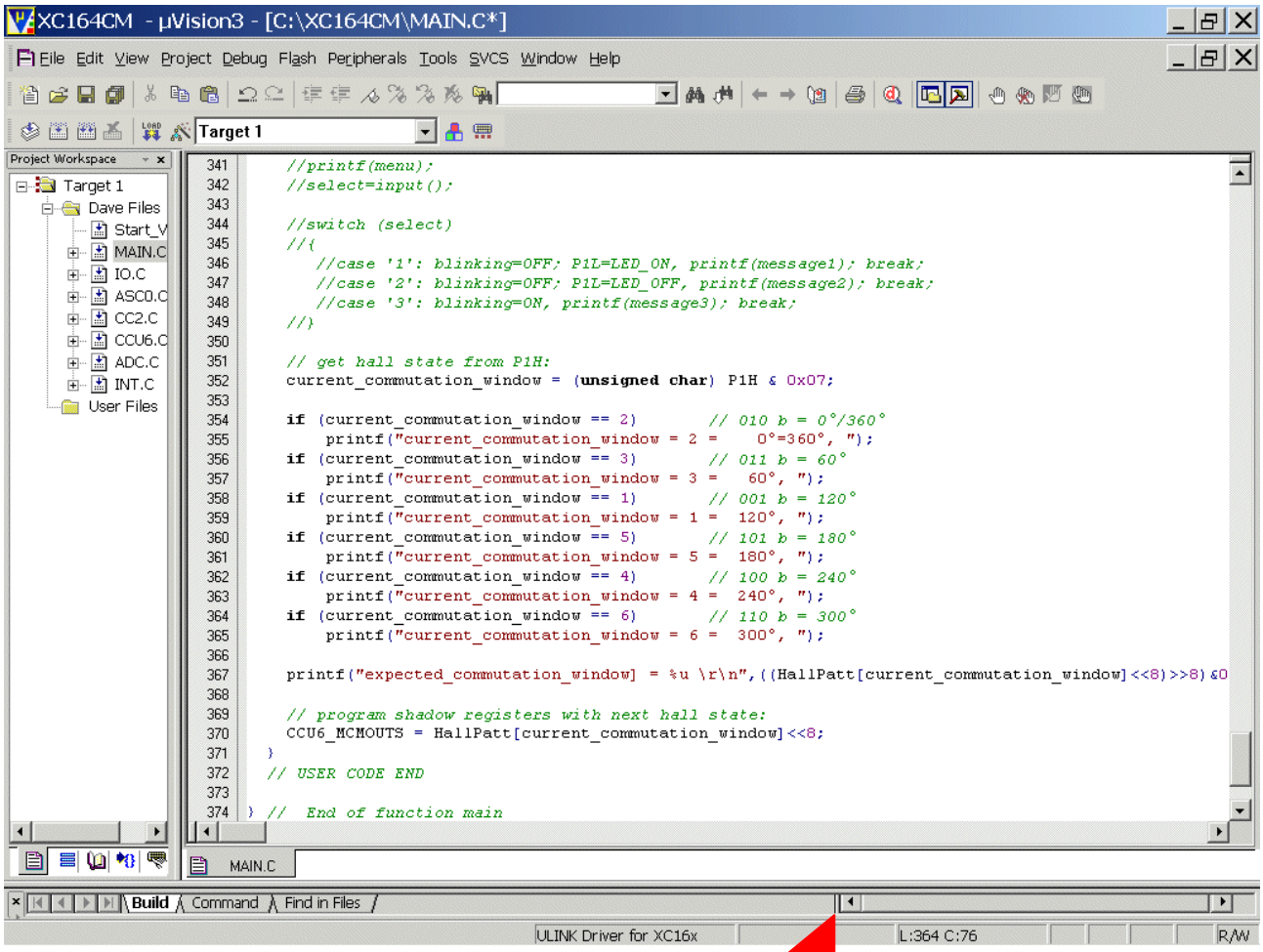
to:

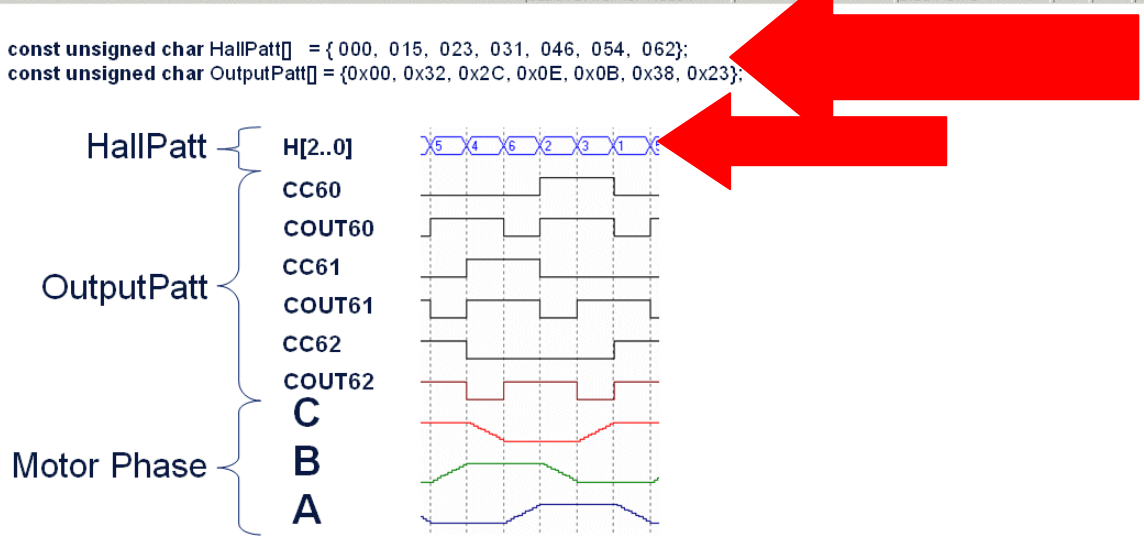
```
    if (current_commutation_window == 2)       // 010 b = 0°/360°
        printf("current_commutation_window = 2 =    0°=360°, ");
    if (current_commutation_window == 3)       // 011 b = 60°
        printf("current_commutation_window = 3 =   60°, ");
    if (current_commutation_window == 1)       // 001 b = 120°
        printf("current_commutation_window = 1 =  120°, ");
    if (current_commutation_window == 5)       // 101 b = 180°
        printf("current_commutation_window = 5 =  180°, ");
    if (current_commutation_window == 4)       // 100 b = 240°
        printf("current_commutation_window = 4 =  240°, ");
    if (current_commutation_window == 6)       // 110 b = 300°
        printf("current_commutation_window = 6 =  300°, ");
```


XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A66
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
339 while (1)
340 {
341 //printf(menu);
342 //select=input();
343
344 //switch (select)
345 //{
346 //case '1': blinking=OFF; P1L=LED_ON, printf(message1); break;
347 //case '2': blinking=OFF; P1L=LED_OFF, printf(message2); break;
348 //case '3': blinking=ON, printf(message3); break;
349 //}
350
351 // get hall state from P1H:
352 current_commutation_window = (unsigned char) P1H & 0x07;
353
354 if (current_commutation_window == 2) // 010 b = 0°/360°
355 printf("current_commutation_window = 2 = 0°=360°, ");
356 if (current_commutation_window == 3) // 011 b = 60°
357 printf("current_commutation_window = 3 = 60°, ");
358 if (current_commutation_window == 1) // 001 b = 120°
359 printf("current_commutation_window = 1 = 120°, ");
360 if (current_commutation_window == 5) // 101 b = 180°
361 printf("current_commutation_window = 5 = 180°, ");
362 if (current_commutation_window == 4) // 100 b = 240°
363 printf("current_commutation_window = 4 = 240°, ");
364 if (current_commutation_window == 6) // 110 b = 300°
365 printf("current_commutation_window = 6 = 300°, ");
366
367 }
368 // USER CODE END
369
370 } // End of function main
MAIN.C
Build Command Find in Files
For Help, press F1
ULINK Driver for XC16x
L:354 C:5
R/W

Double click MAIN.C and insert Code:

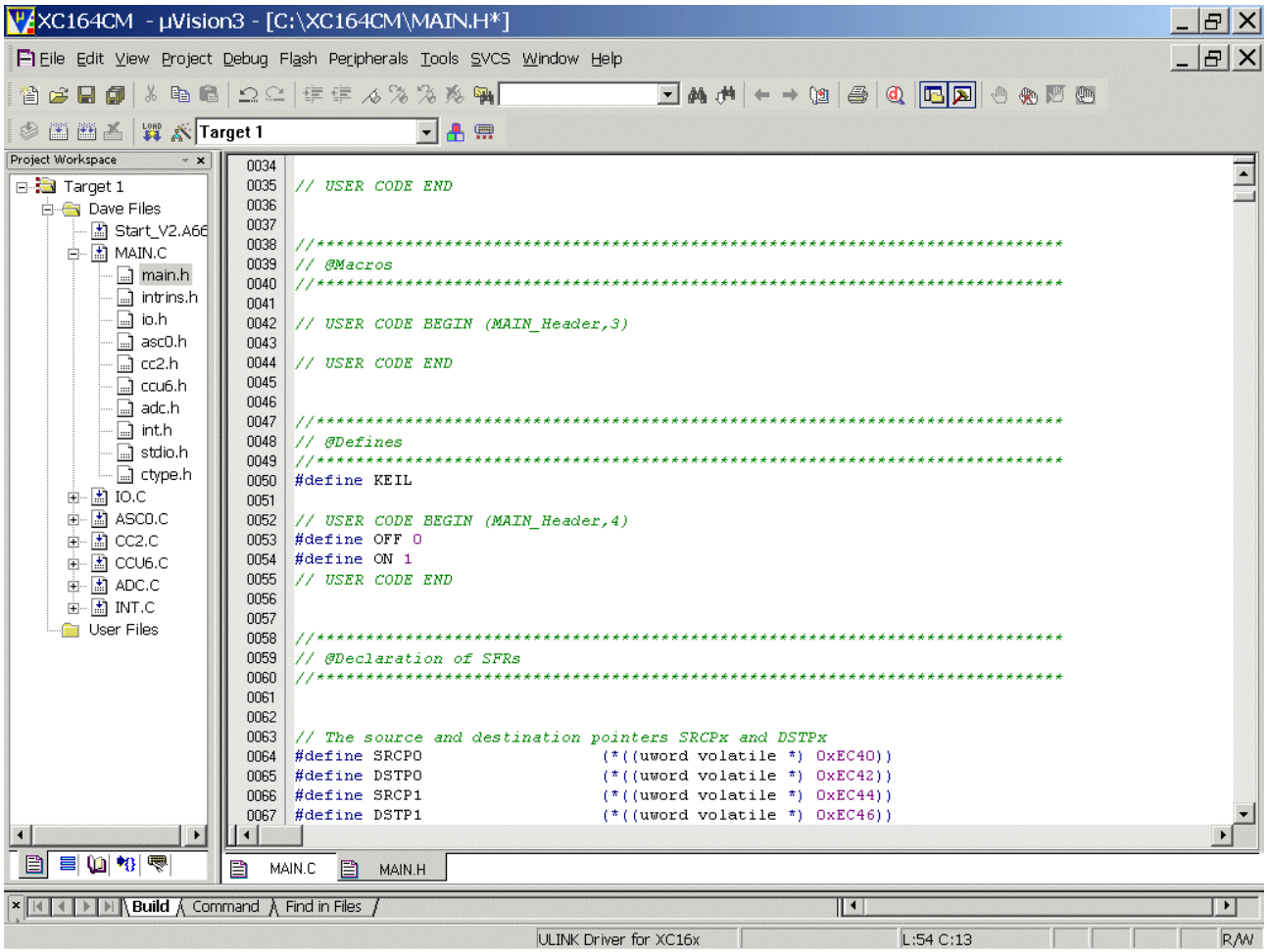
```
printf("expected_commutation_window] = %u
\r\n",((HallPatt[current_commutation_window]<<8)>>8)&0x07);

// program shadow registers with next hall state:
CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;
```





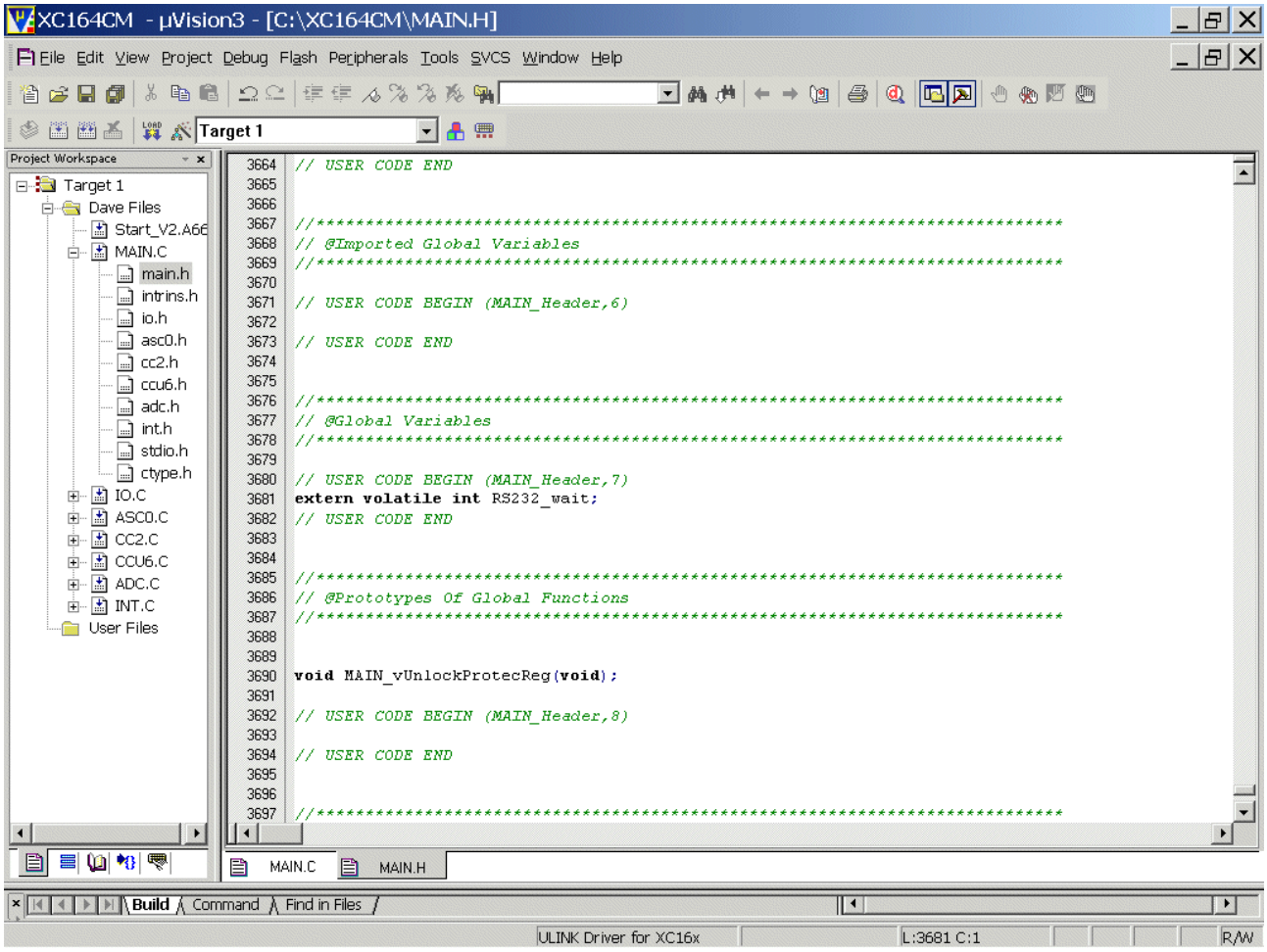
Double click main.h and DELETE Defines:

```
#define LED_ON 0x0000
#define LED_OFF 0xFFFF
```



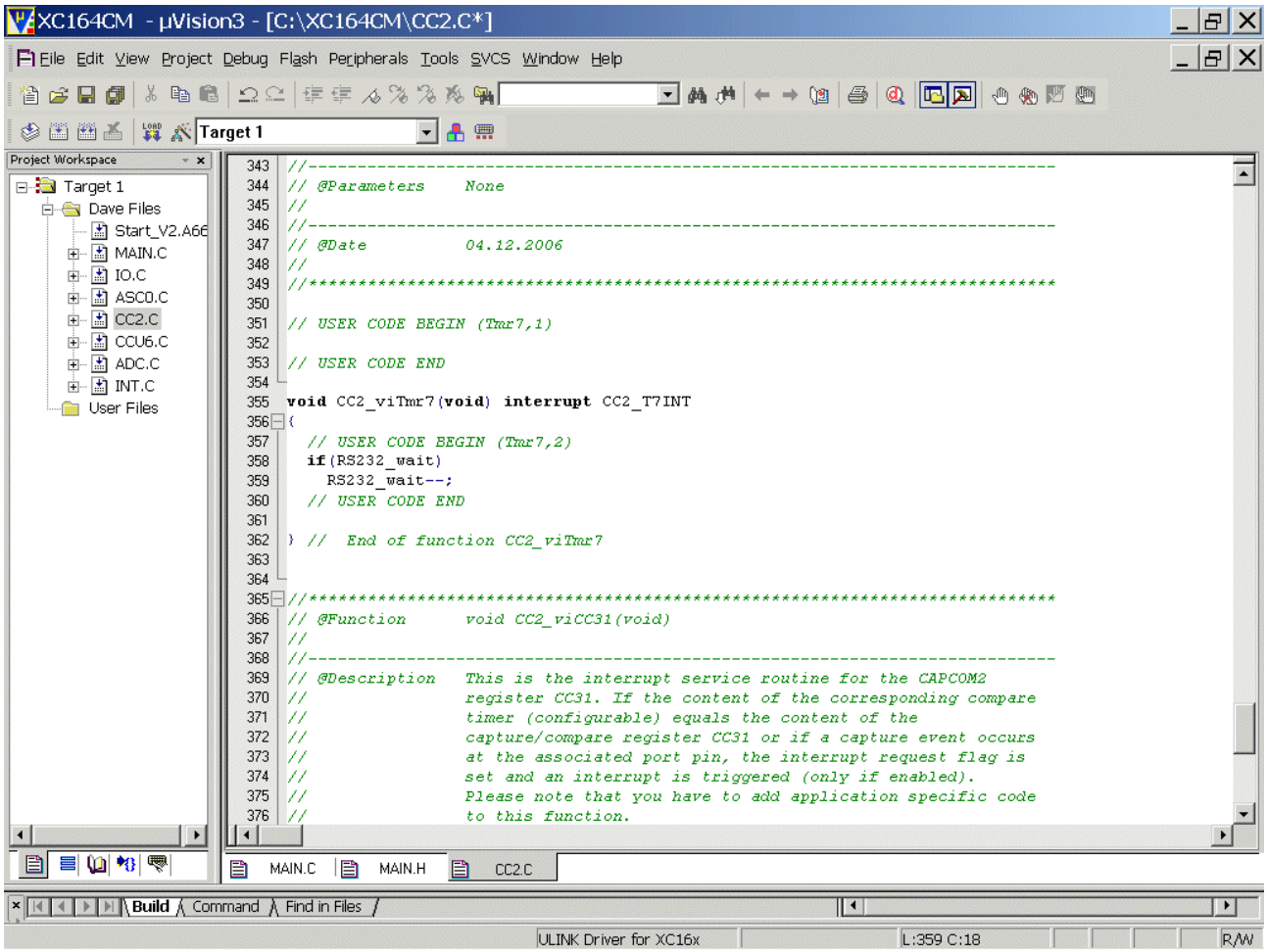
Double click main.h and DELETE Global Variable:

```
extern bit blinking;
```

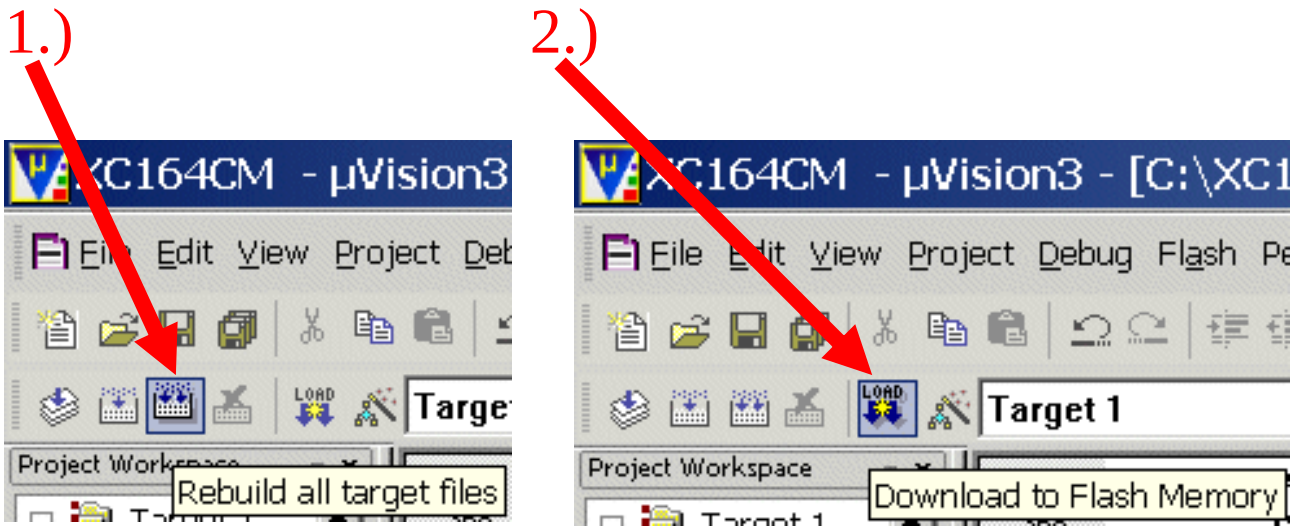


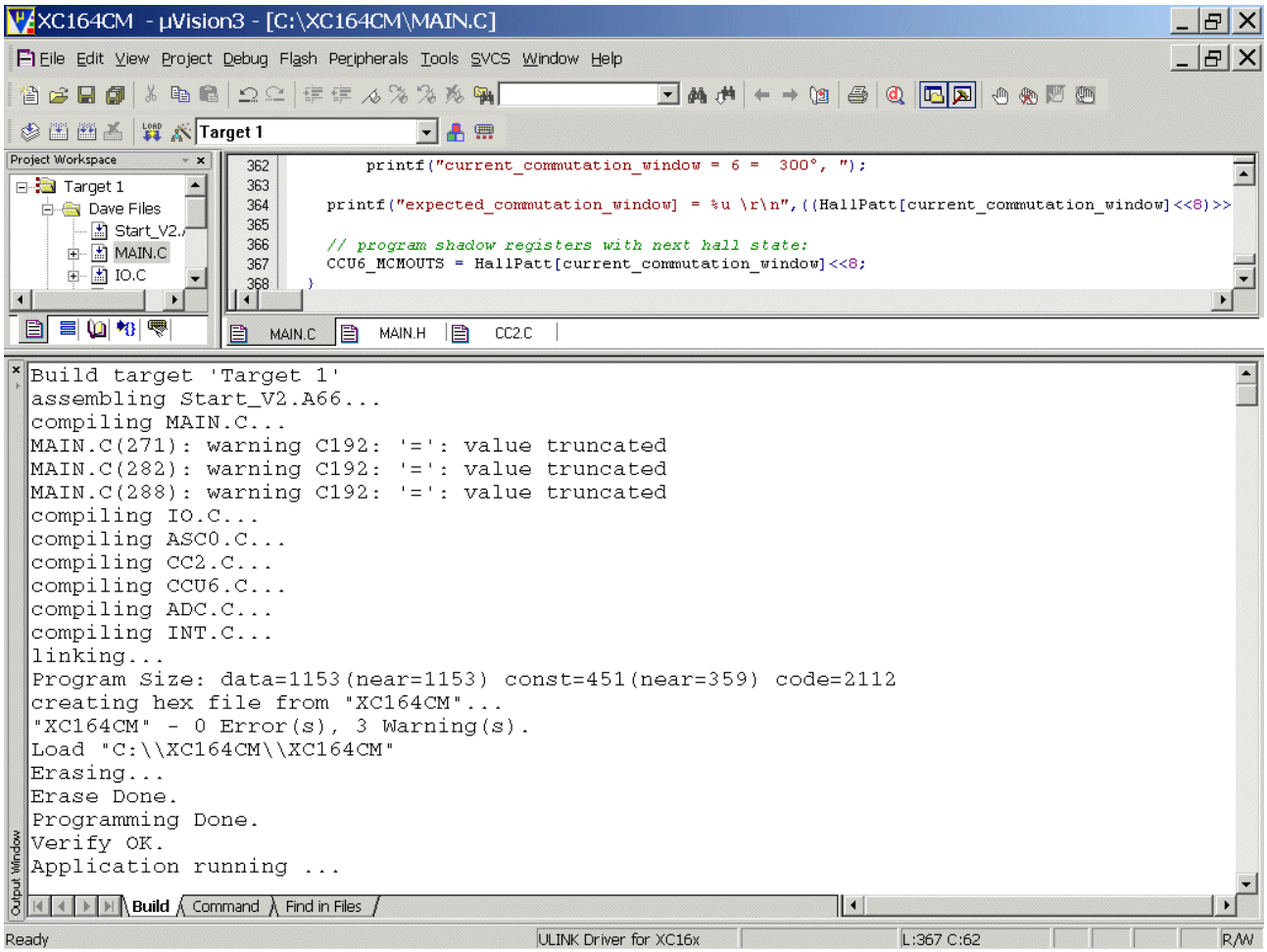
Double click CC2.C and DELETE Code:

```
if(blinking)
   P1L=P1L^0xFFFF;
```



Build Application:

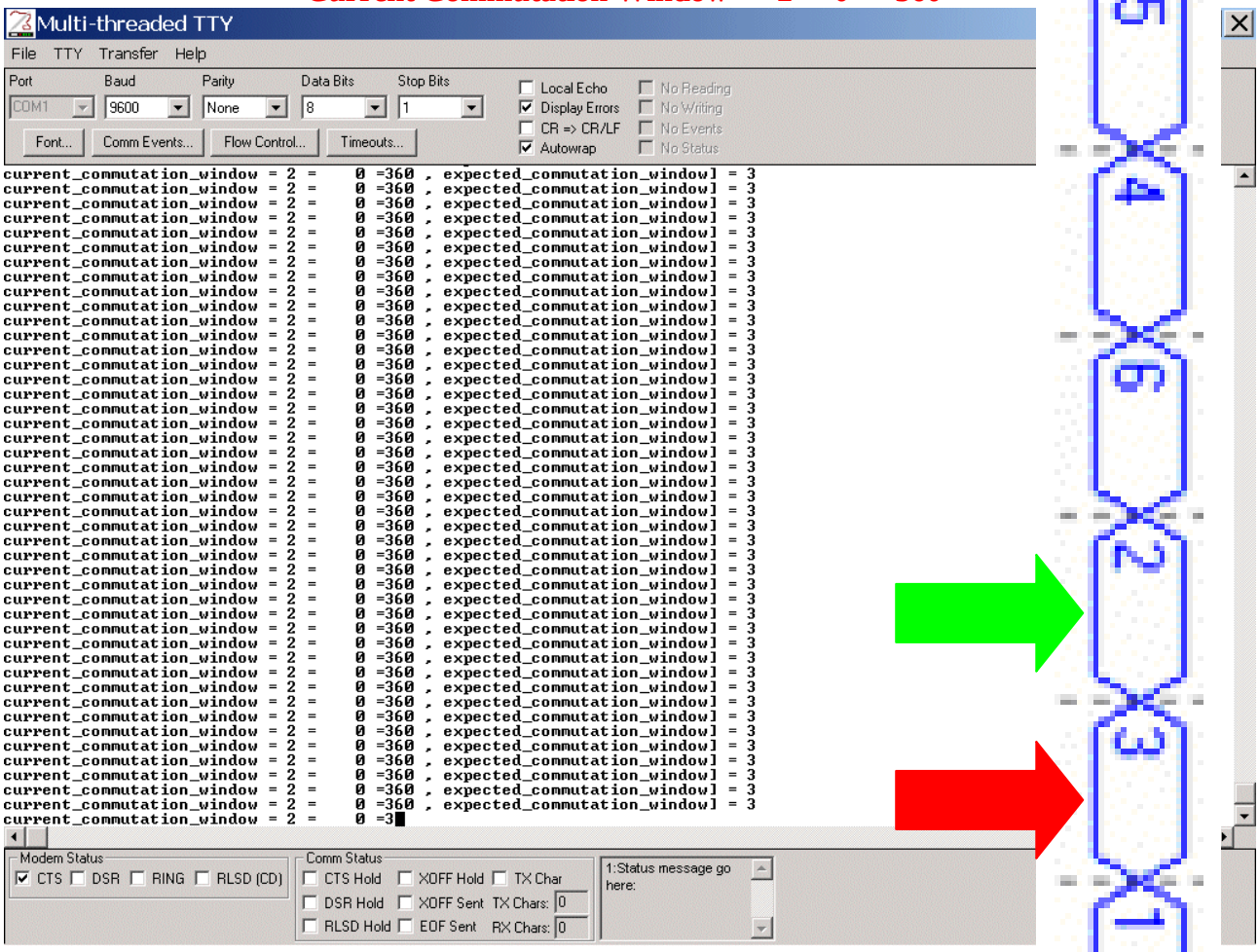




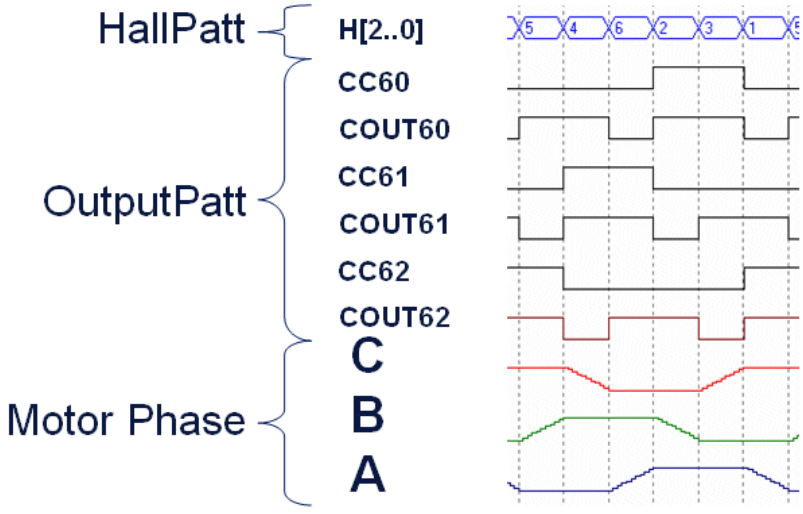
And see the result:

Note: The motor has to be turned by hand ☺.

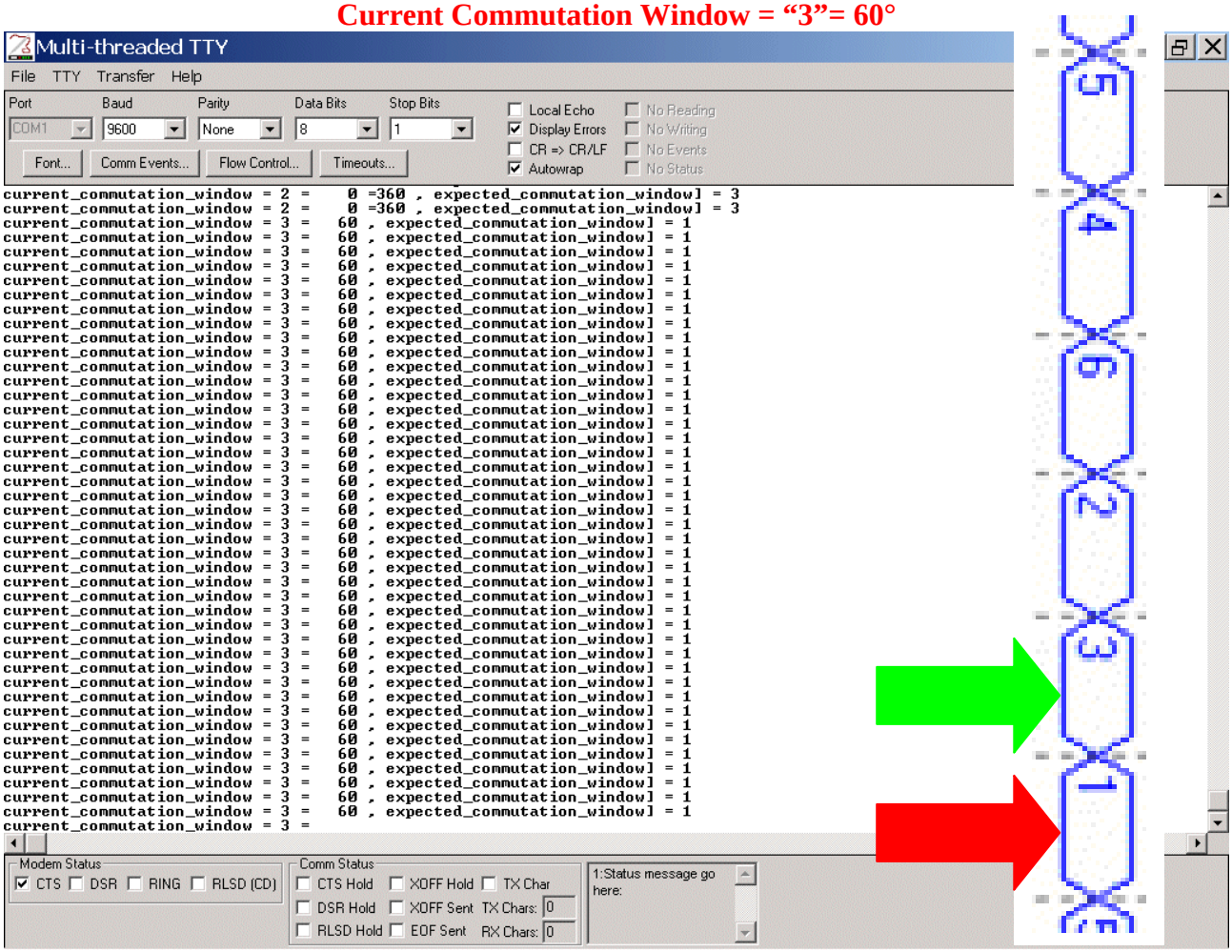
Current Commutation Window = "2"= 0° = 360°



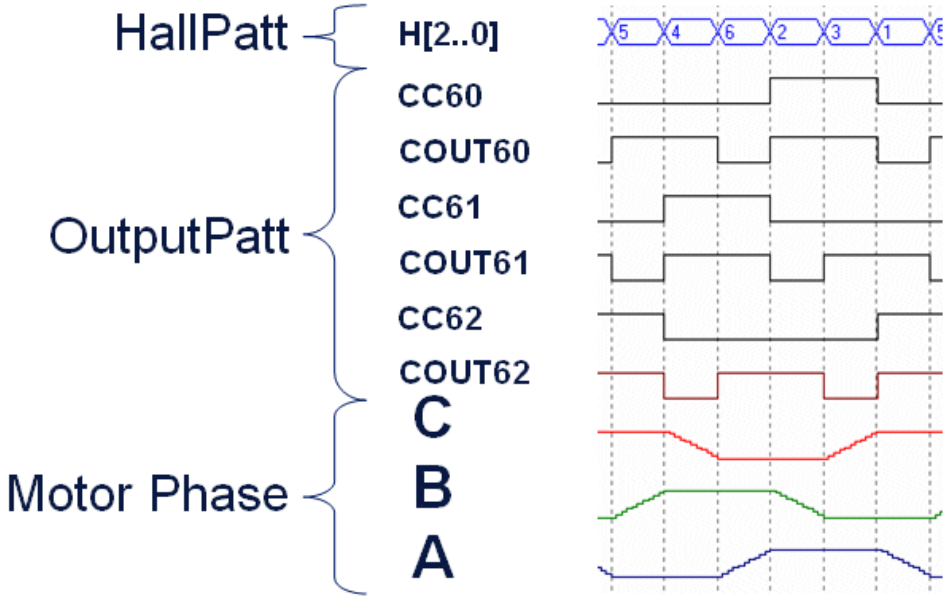

```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x2
```



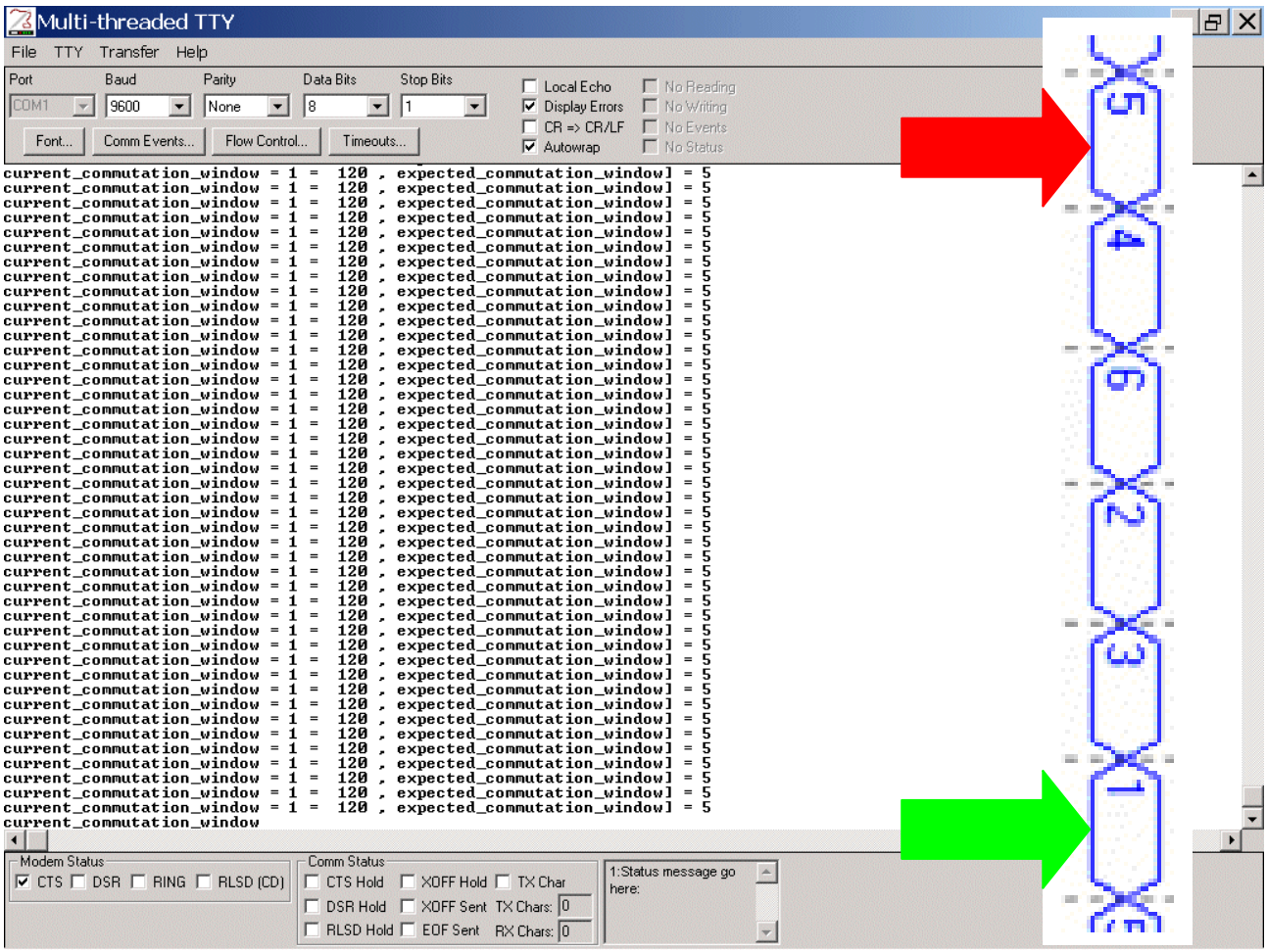
Current Commutation Window = "3"= 60°



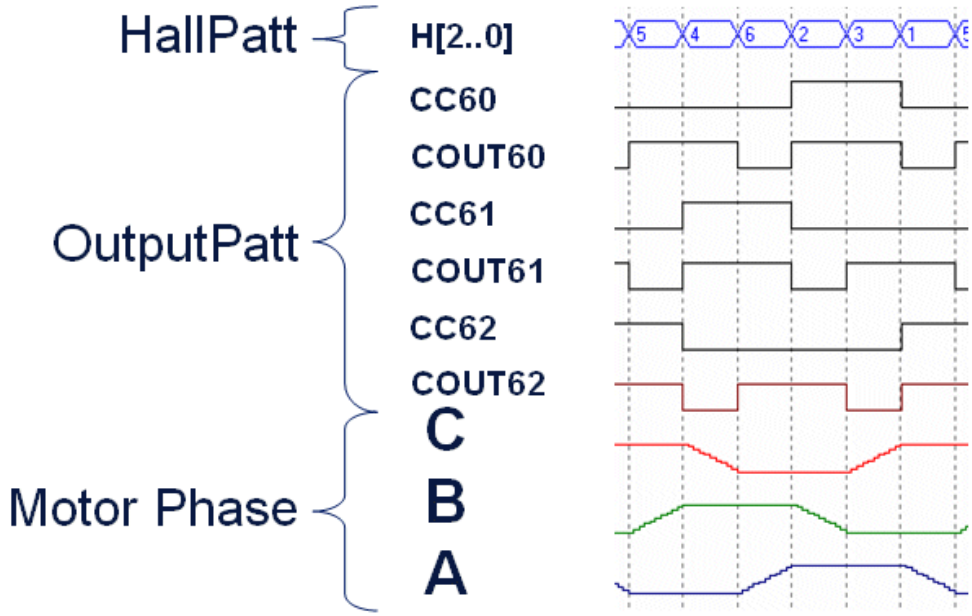

```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```



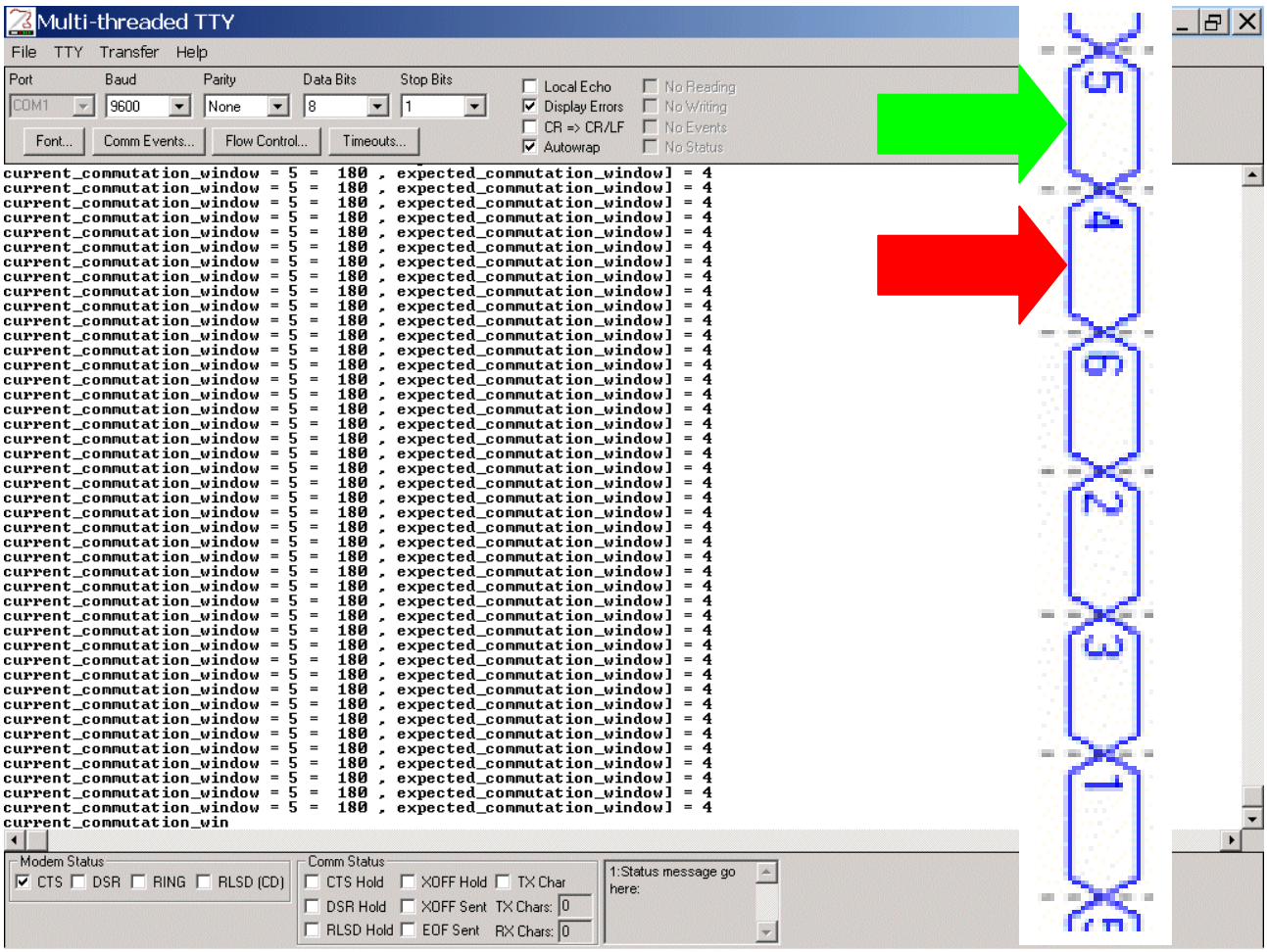
Current Commutation Window = “1”= 120°



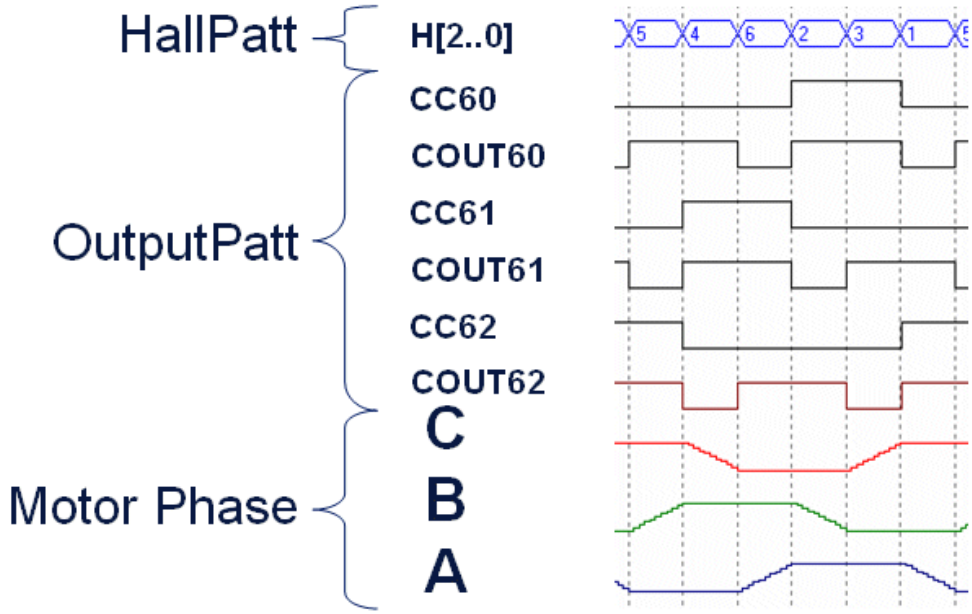

```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```



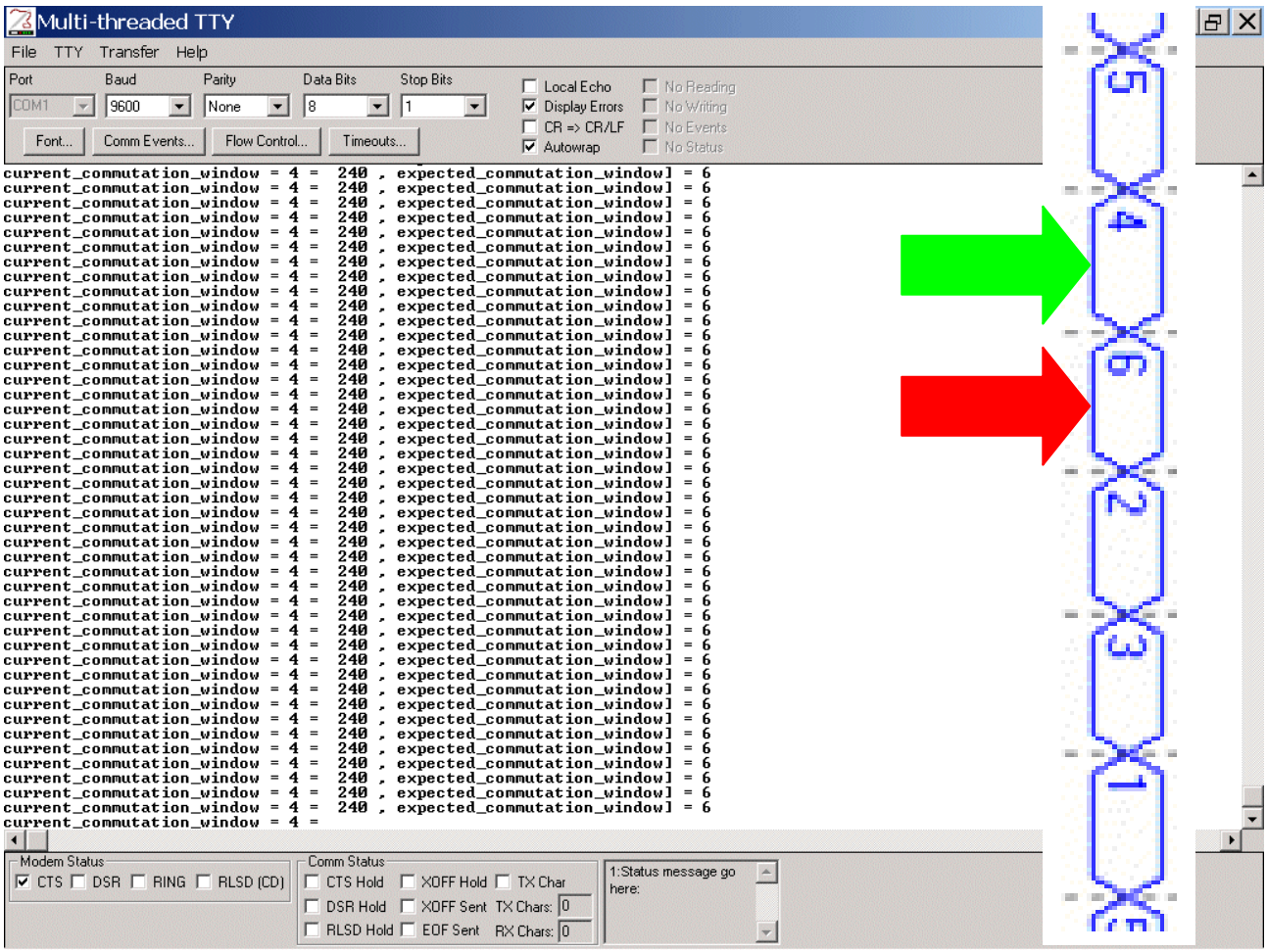
Current Commutation Window = “5”= 180°



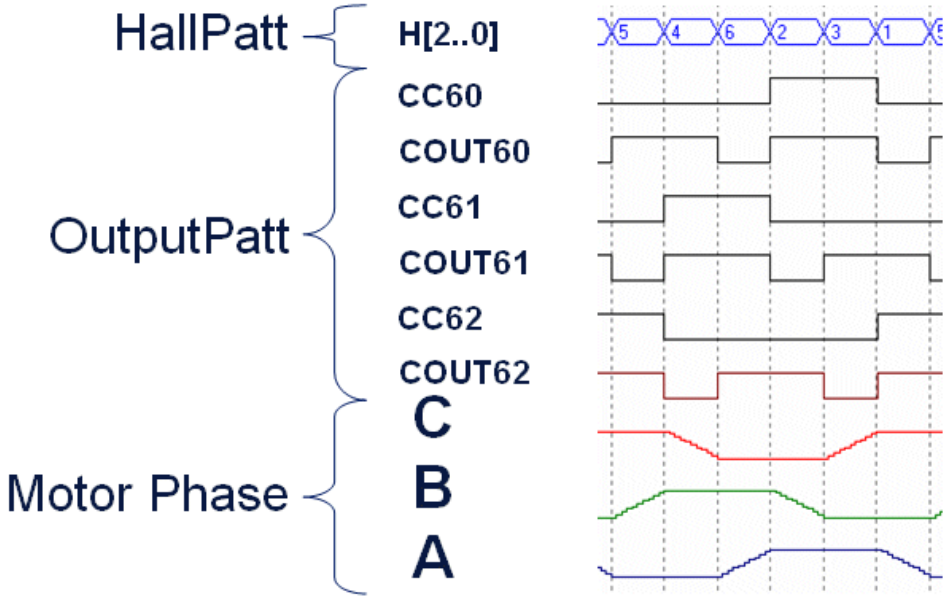

```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```



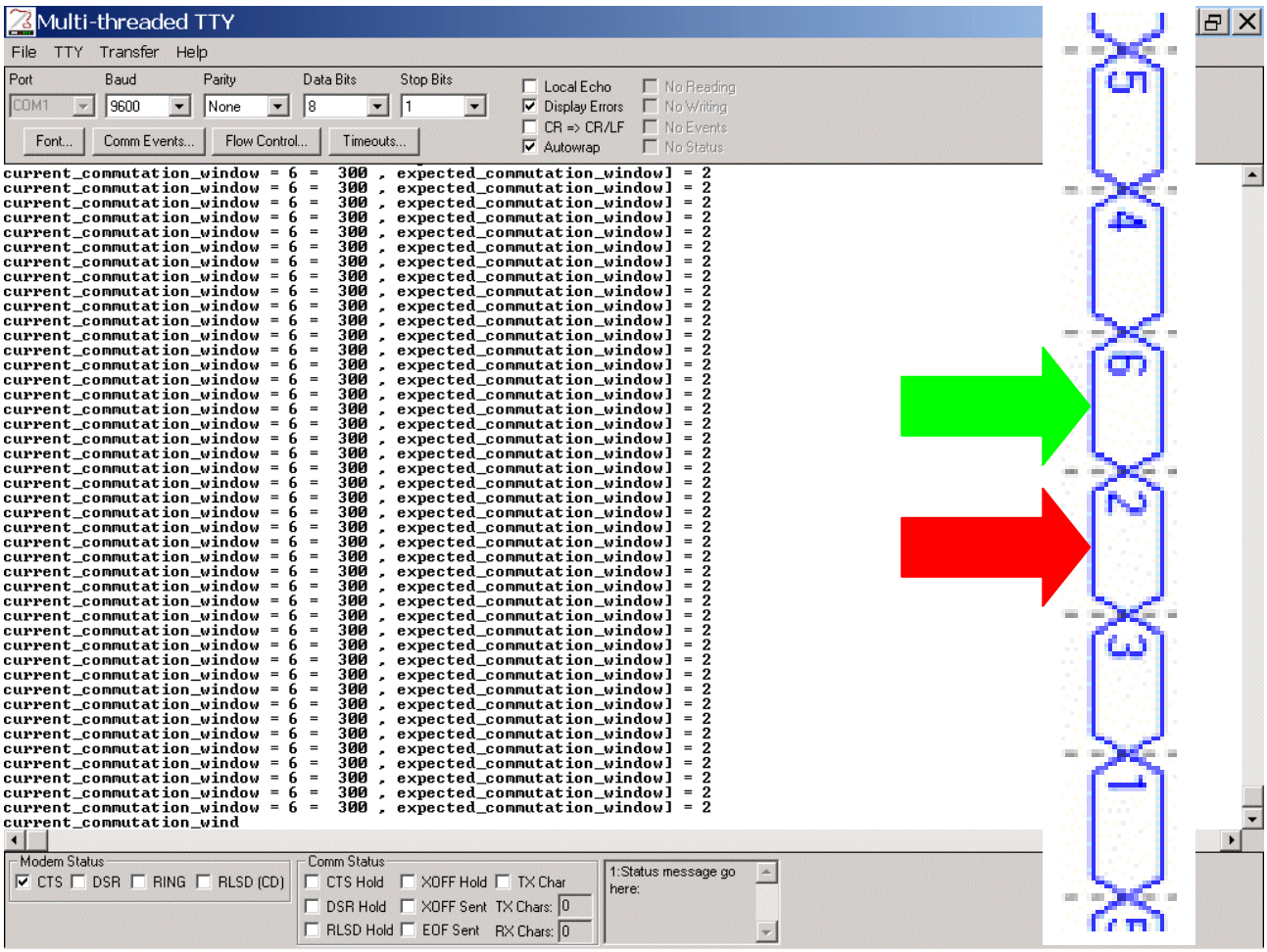
Current Commutation Window = "4"= 240°



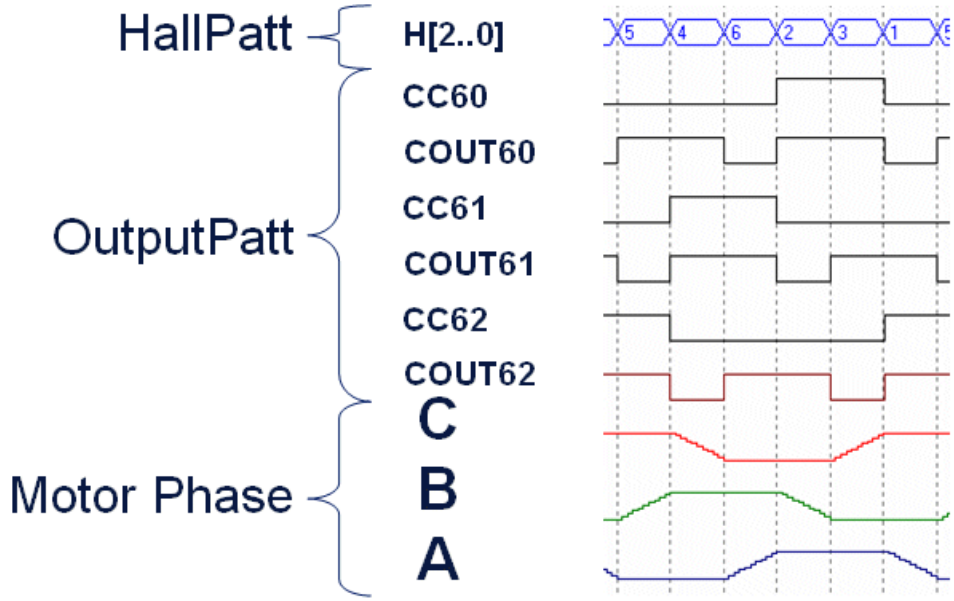

```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```



Current Commutation Window = “6”= 300°




```
const unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```

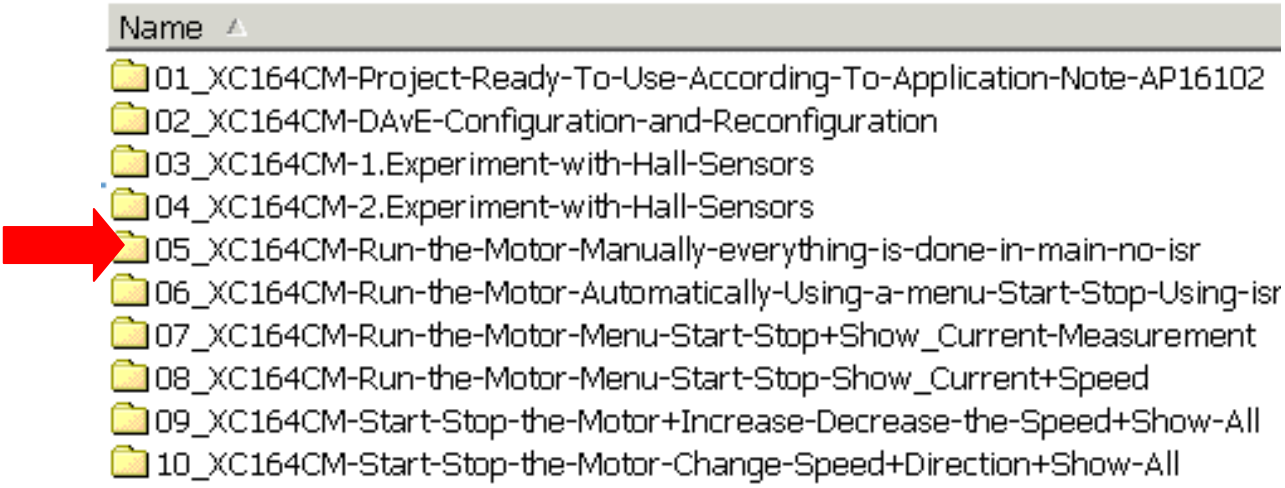


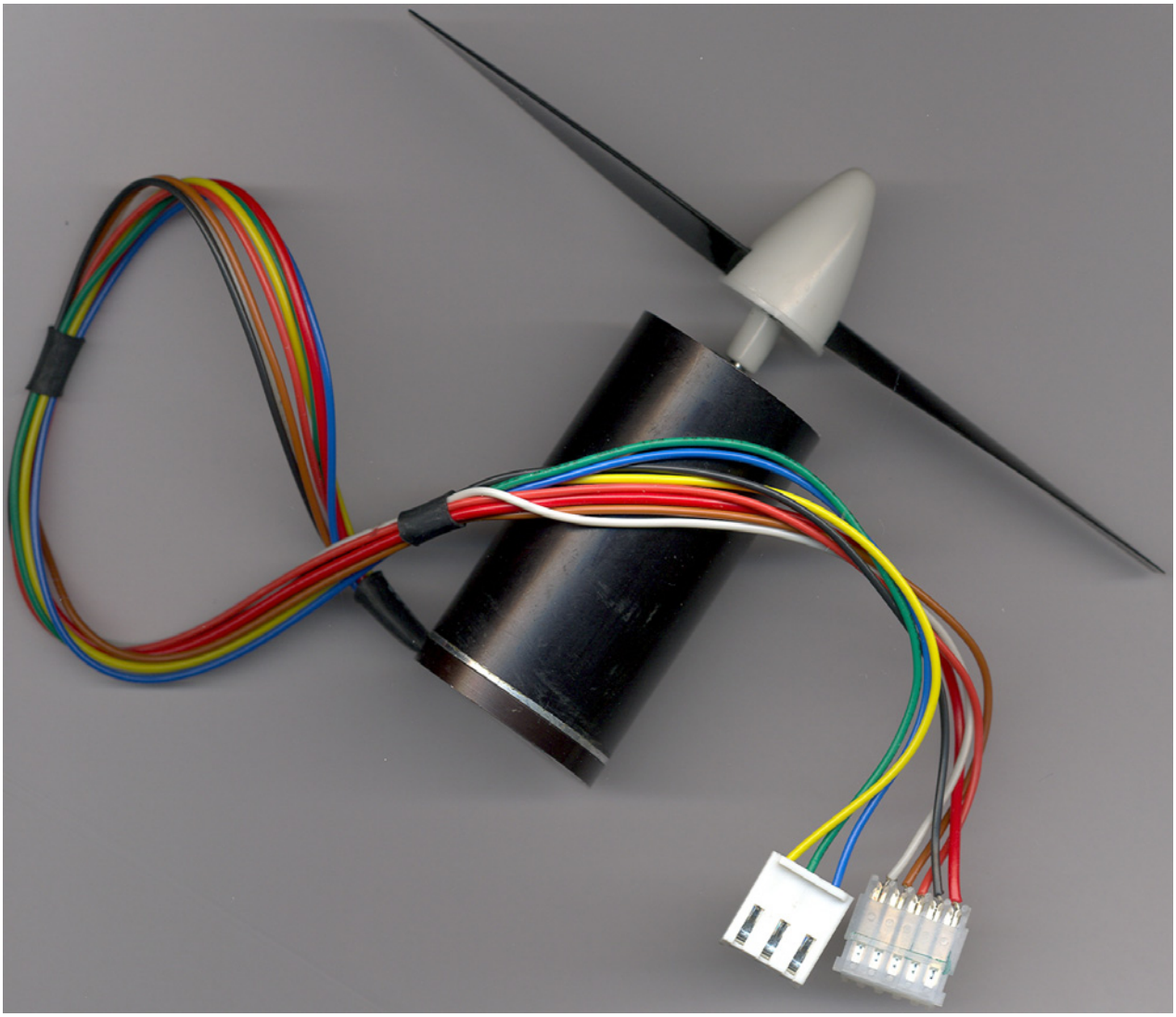
5.)

Running the Motor Manually

[everything is done in void main (void) without any interrupt-function] :

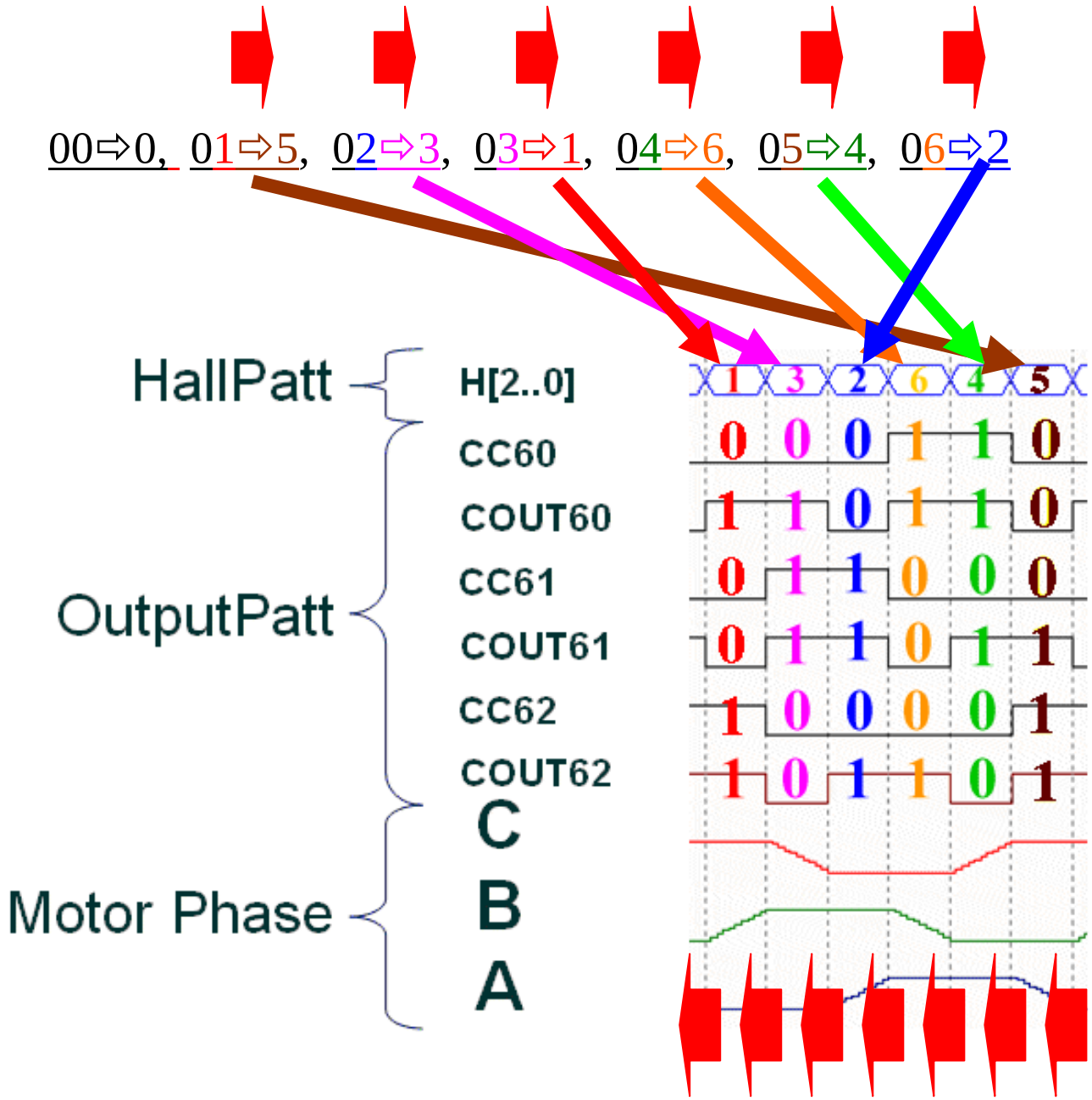
Relevant Project:

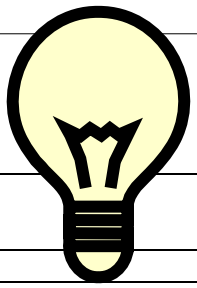




Register Information / Additional Information:

```
// State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt[]   = { 000,  015,  023,  031,  046,  054,  062 };
```





Register Information / Additional Information:

```
// State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt[]   = { 000,  015,  023,  031,  046,  054,  062 };
```

```
// program shadow registers with next hall state
CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;
```

CCU6_MCMOUTS
Multi-Ch. Mode Outp. Shad. R. SFR (E8CA$_H$/--) Reset Value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STR HP	-	CURHS			EXPHS			STR MCM	-	MCMPS					
w	-	rw			rw			w	-	rw					

CURHS	[13:11]	rw	**Current Hall Pattern Shadow Field** CURHS is the shadow field for bitfield CURH. The bitfield is transferred to CURH when a correct Hall event is detected.
EXPHS	[10:8]	rw	**Expected Hall Pattern Shadow Field** EXPHS is the shadow field for bitfield EXPH. The bitfield is transferred to EXPH when a correct Hall event is detected.

CCU6_MCMOUT
Multi-Ch. Mode Output Reg. SFR (E8CC$_H$/--) Reset Value: 0000$_H$

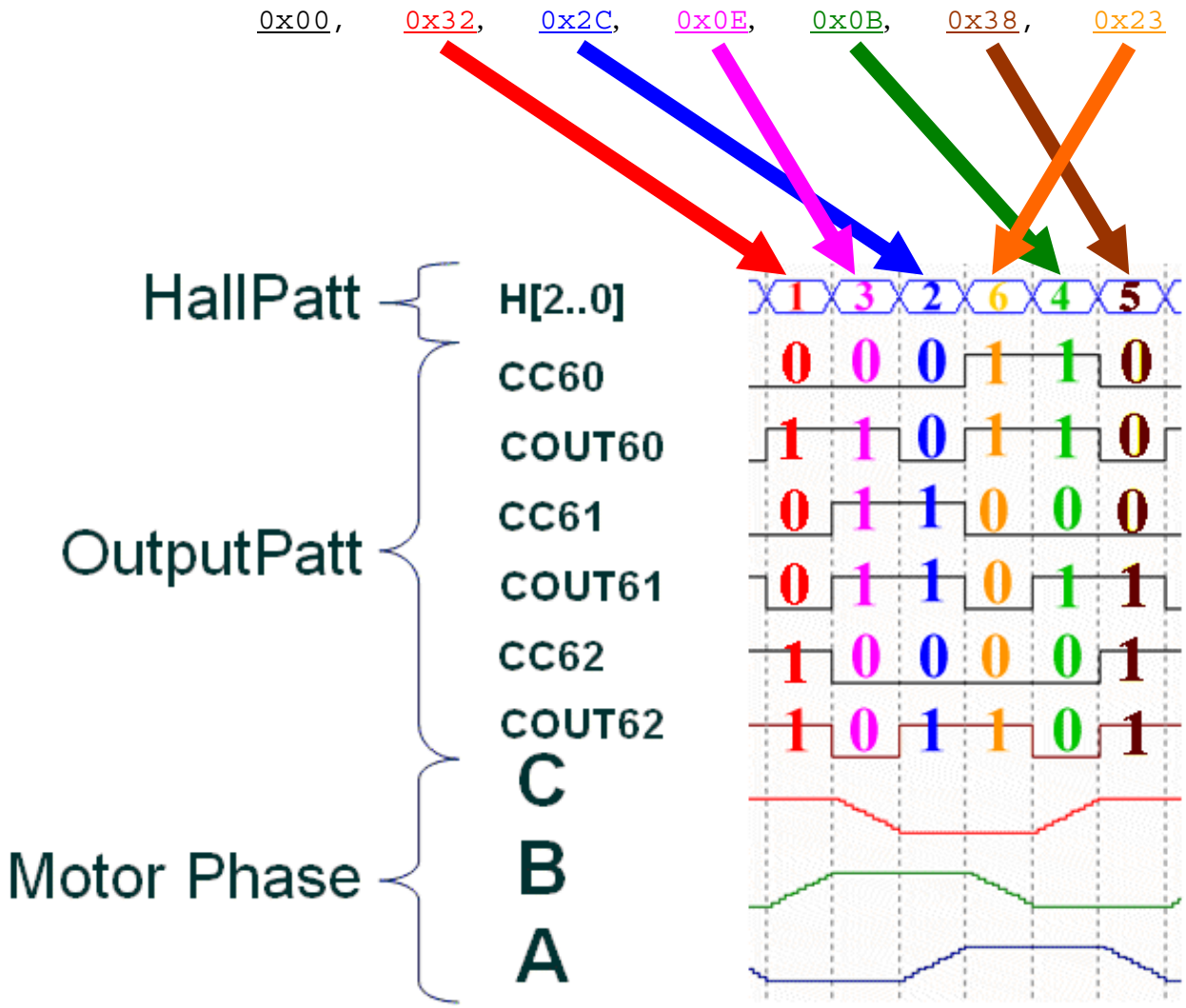
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	CURH			EXPH			-	R	MCMP					
-	-	rh			rh			-	rh	rh					

Field	Bits	Type	Description
CURH[1)]	[13:11]	rh	**Current Hall Pattern** CURH is written by a shadow transfer from bitfield CURHS. Bitfield CURH is compared to the sampled Hall pattern after every detected edge at the Hall sensor inputs CC6POSx. If the pattern match, the detected edge has been an invalid transition (e.g. due to a spike), and no further action is performed. If the sampled pattern do not either match CURH or EXPH, a Wrong Hall Event signal is set, which can trigger further actions.
EXPH[1)]	[10:8]	rh	**Expected Hall Pattern** EXPH is written by a shadow transfer from bitfield EXPHS. Bitfield EXPH is compared to the sampled Hall pattern after every detected edge at the Hall sensor inputs CC6POSx. If the pattern match, a Correct Hall Event signal is generated, which triggers further actions.

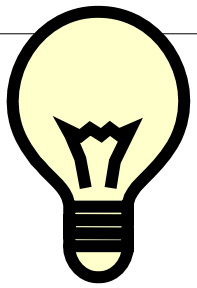
Register Information / Additional Information:

```
// State-Transition-Table - next output pattern (Hex)
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B,
0x38, 0x23};
```

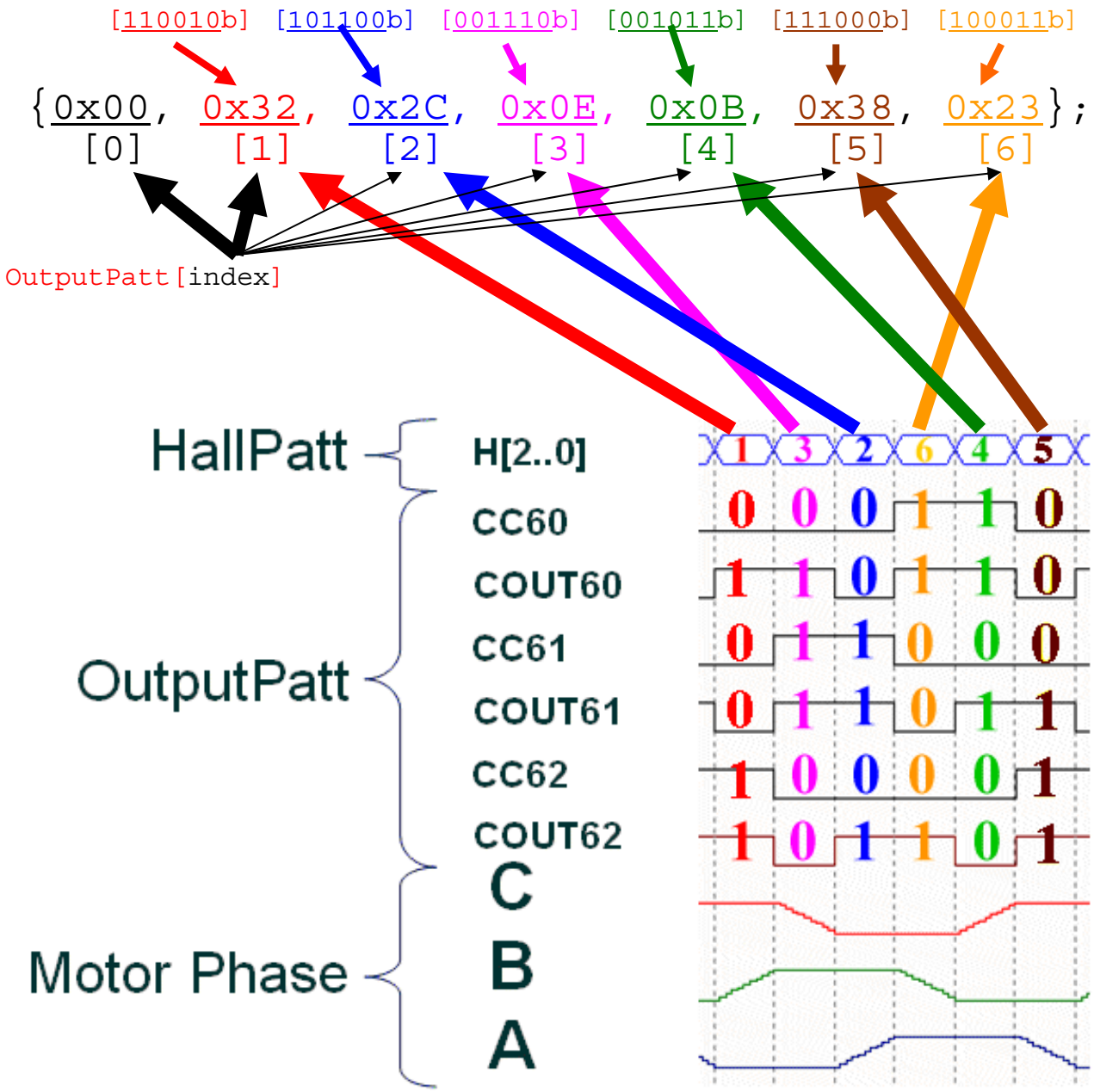
```
// program shadow registers with next output pattern:
CCU6 MCMOUTS |= OutputPatt[current commutation window];
```

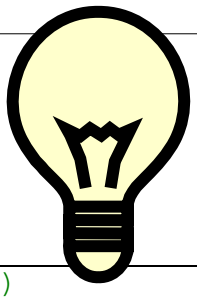


Register Information / Additional Information:



```
// State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B,
0x38, 0x23};
```





Register Information / Additional Information:

```
// State-Transition-Table - next output pattern (Hex)
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B,
0x38, 0x23};
```

```
// program shadow registers with next output pattern:
CCU6 MCMOUTS |= OutputPatt[current commutation window];
```

CCU6_MCMOUTS
Multi-Ch. Mode Outp. Shad. R. XSFR (E8CA$_H$/--) Reset Value: 0000$_H$

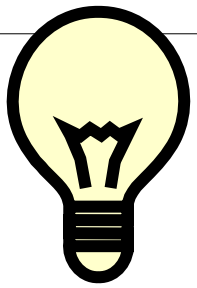
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STRHP	-	CURHS			EXPHS			STRMCM	-	MCMPS					
w	-	rw			rw			w	-	rw					

MCMPS	[5:0]	rw	**Multi-Channel PWM Pattern Shadow Field** MCMPS is the shadow field for bitfield MCMP. The Multi-Channel shadow transfer is triggered according to the transfer conditions defined by register MCMCTR.

CCU6_MCMOUT
Multi-Ch. Mode Output Reg. XSFR (E8CC$_H$/--) Reset Value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	CURH			EXPH			-	R	MCMP					
-	-	rh			rh			-	rh	rh					

MCMP[2)]	[5:0]	rh	**Multi-Channel Modulation Pattern** MCMP contains the output modulation pattern for the Multi-Channel mode, which can set the corresponding output to the passive state. It is written by a shadow transfer from bitfield MCMPS. 0 The output is set to the passive state. 1 The output can deliver the PWM generated by T12 or T13 (according to register MODCTR). MCMP[5:0] corresponds to (left to right): COUT62, CC62, COUT61, CC61, COUT60, CC60.



Register Information / Additional Information:

```
// for this programming example we will do the shadow transfer
always manually
CCU6 MCMOUTS |= 0x8080;
```

C 6_MCMOUTS
M i-Ch. Mode Outp. Shad. R. XSFR (E A$_H$/--) **Reset Value: 0000$_H$**

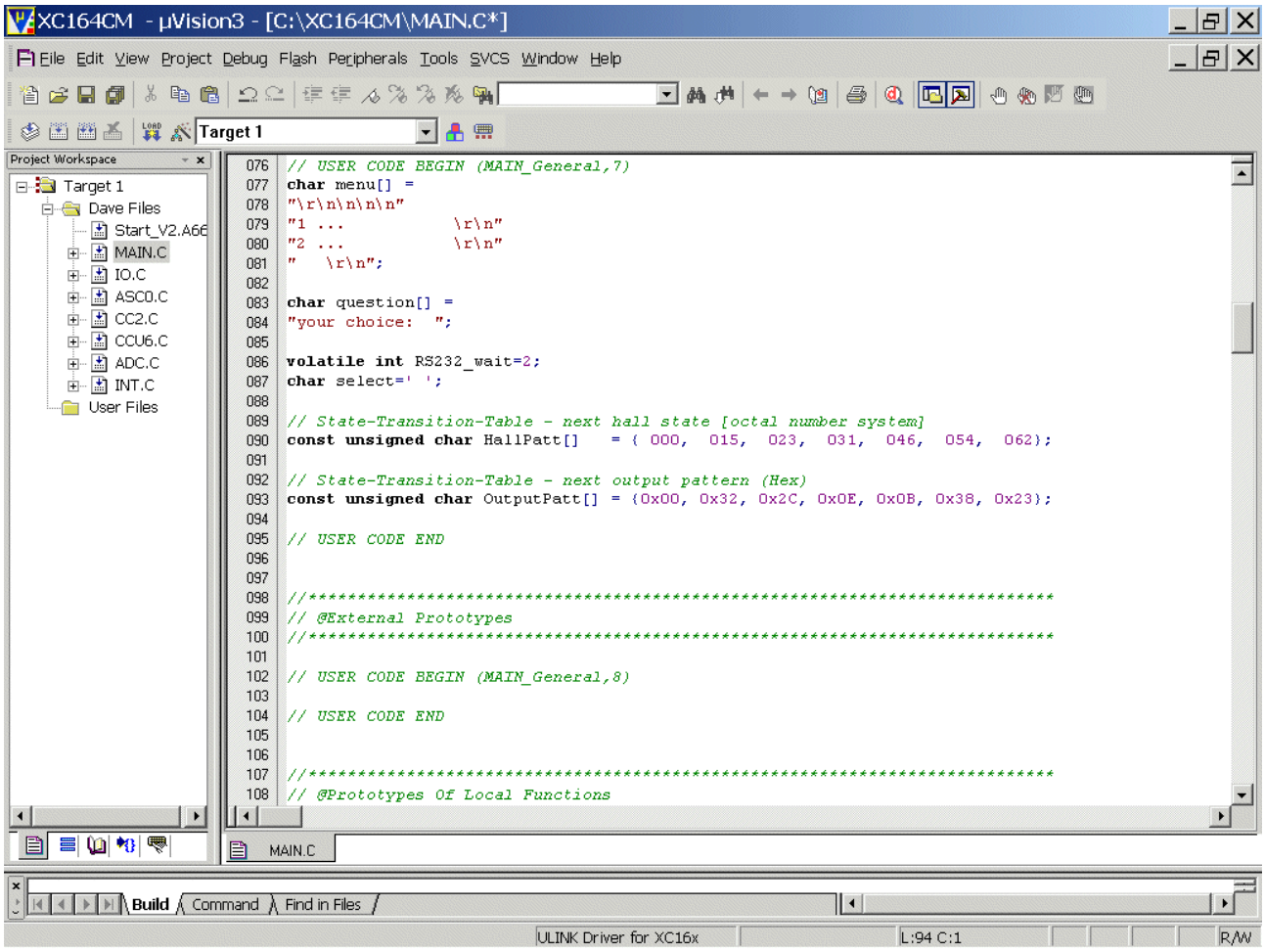
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STR HP	-	CURHS			EXPHS			STR MCM	-	MCMPS					
w	-	rw			rw			w	-	rw					

Field	Bits	Type	Description
STRHP	15	w	**Shadow Transfer Request for the Hall Patterns** Setting this bit during a write action leads to an immediate update of bitfields CURH and EXPH by the value written to CURHS and EXPHS. This functionality permits an update triggered by SW. When read, this bit always delivers 0. 0 CURH and EXPH are updated according to the defined HW action. The write access to CURHS and EXPHS does not modify bitfields CURH and EXPH. 1 CURH and EXPH are updated by the value written to bitfields CURHS and EXPHS.

Field	Bits	Type	Description
STRMCM	7	w	**Shadow Transfer Request for MCMPS** Setting this bit during a write action leads to an immediate update of bitfield MCMP by the value written to MCMPS. This functionality permits an update triggered by SW. When read, this bit always delivers 0. 0 MCMP is updated according to the defined HW action. The write access to MCMPS does not modify MCMP. 1 MCMP is updated by the value written to MCMPS.

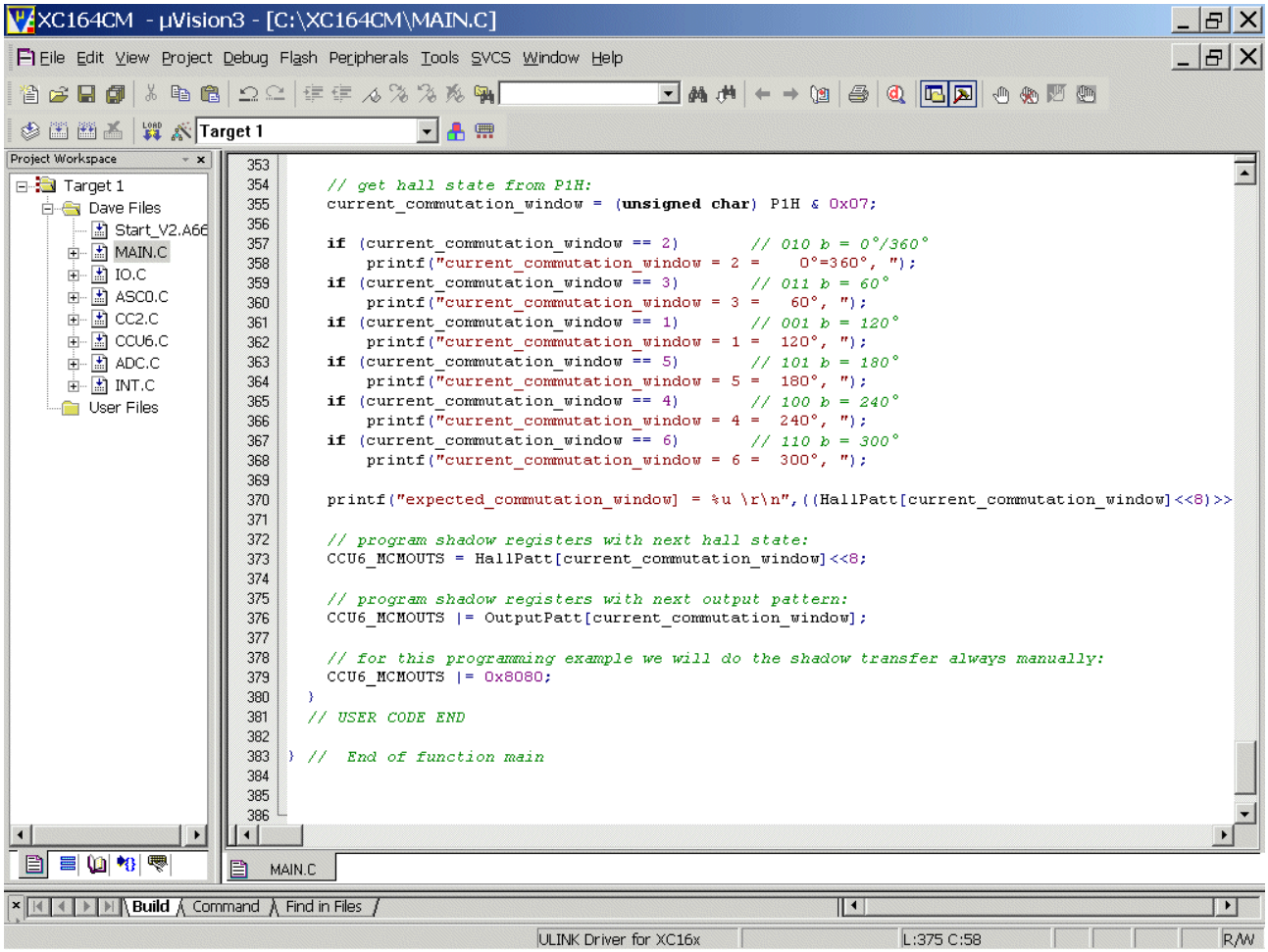
Double click MAIN.C and insert Global Variable:

```
// State-Transition-Table - next output pattern (Hex)
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B,
0x38, 0x23};
```



Double click MAIN.C and insert Code:

```
// program shadow registers with next output pattern:
CCU6_MCMOUTS |= OutputPatt[current_commutation_window];

// for this programming example we will do the shadow transfer
always manually:
CCU6_MCMOUTS |= 0x8080;
```

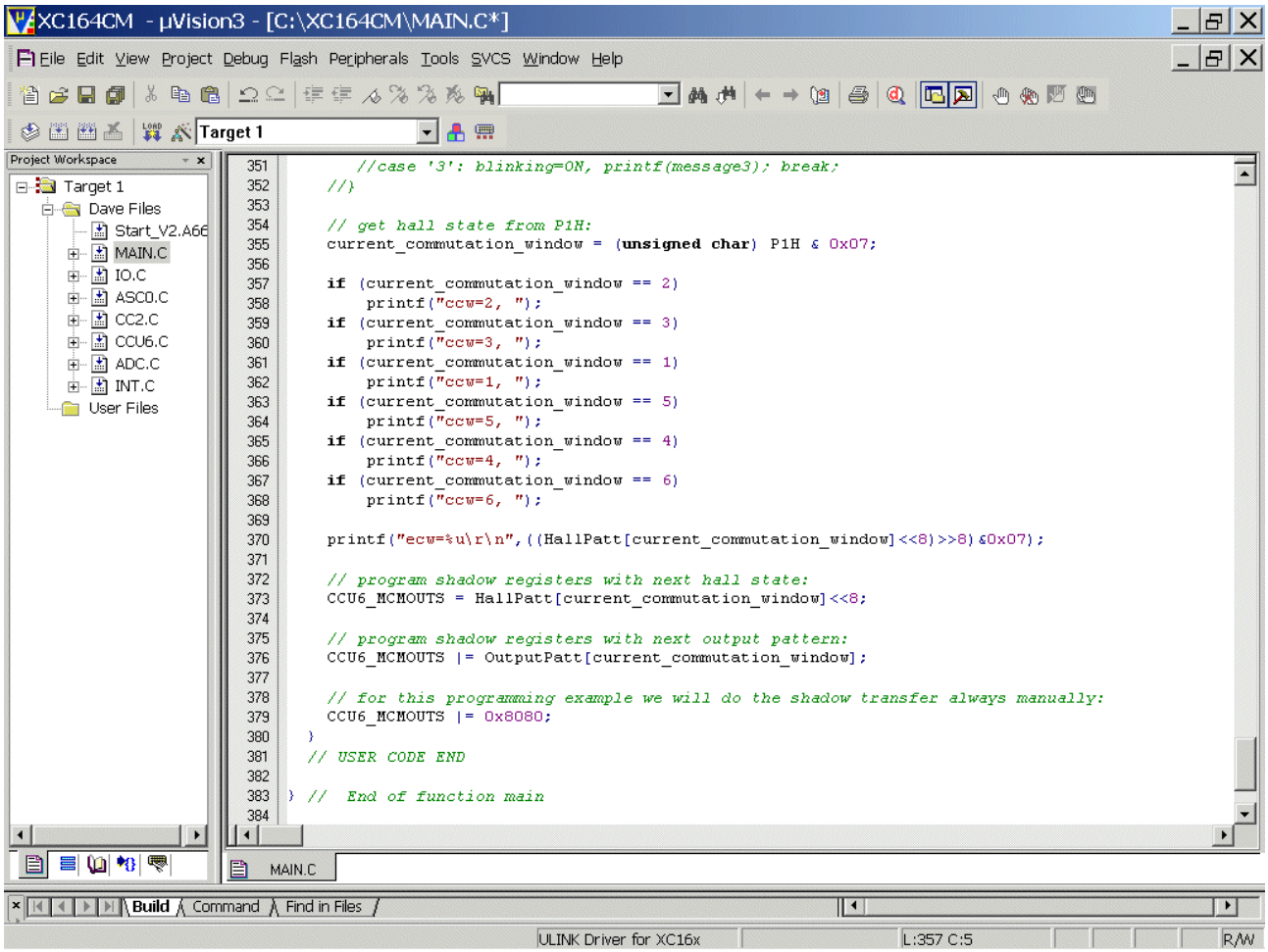


Double click MAIN.C and change Code from:

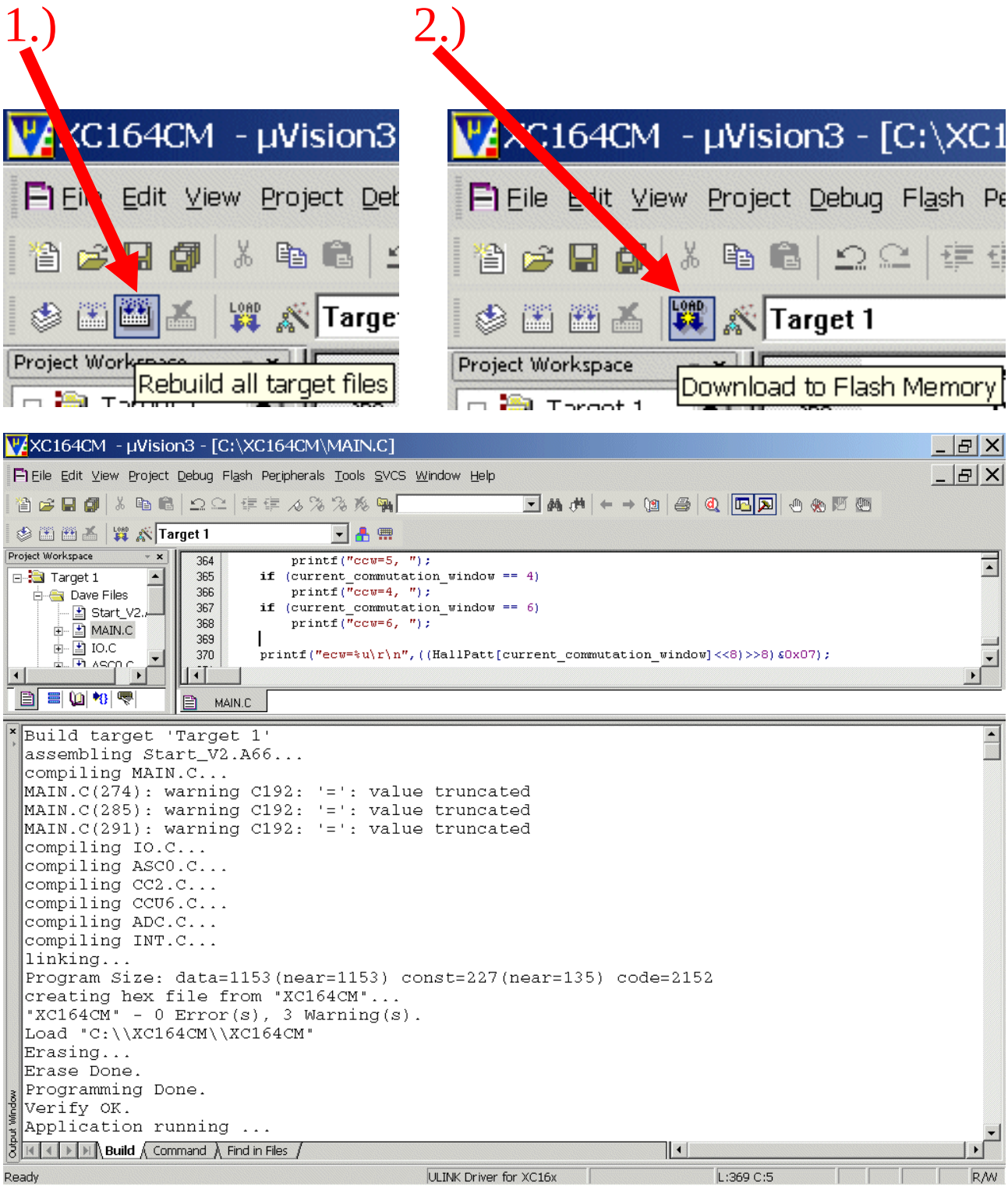
```
if (current_commutation_window == 2)       // 010 b = 0°/360°
    printf("current_commutation_window = 2 =    0°=360°, ");
if (current_commutation_window == 3)       // 011 b = 60°
    printf("current_commutation_window = 3 =   60°, ");
if (current_commutation_window == 1)       // 001 b = 120°
    printf("current_commutation_window = 1 =  120°, ");
if (current_commutation_window == 5)       // 101 b = 180°
    printf("current_commutation_window = 5 =  180°, ");
if (current_commutation_window == 4)       // 100 b = 240°
    printf("current_commutation_window = 4 =  240°, ");
if (current_commutation_window == 6)       // 110 b = 300°
    printf("current_commutation_window = 6 =  300°, ");

printf("expected_commutation_window] = %u
\r\n",((HallPatt[current_commutation_window]<<8)>>8)&0x07);
```

to:

```
    if (current_commutation_window == 2)
        printf("ccw=2, ");
    if (current_commutation_window == 3)
        printf("ccw=3, ");
    if (current_commutation_window == 1)
        printf("ccw=1, ");
    if (current_commutation_window == 5)
        printf("ccw=5, ");
    if (current_commutation_window == 4)
        printf("ccw=4, ");
    if (current_commutation_window == 6)
        printf("ccw=6, ");

    printf("ecw=%u\r\n",((HallPatt[current_commutation_window]<<8)>>8)&0x07);
```

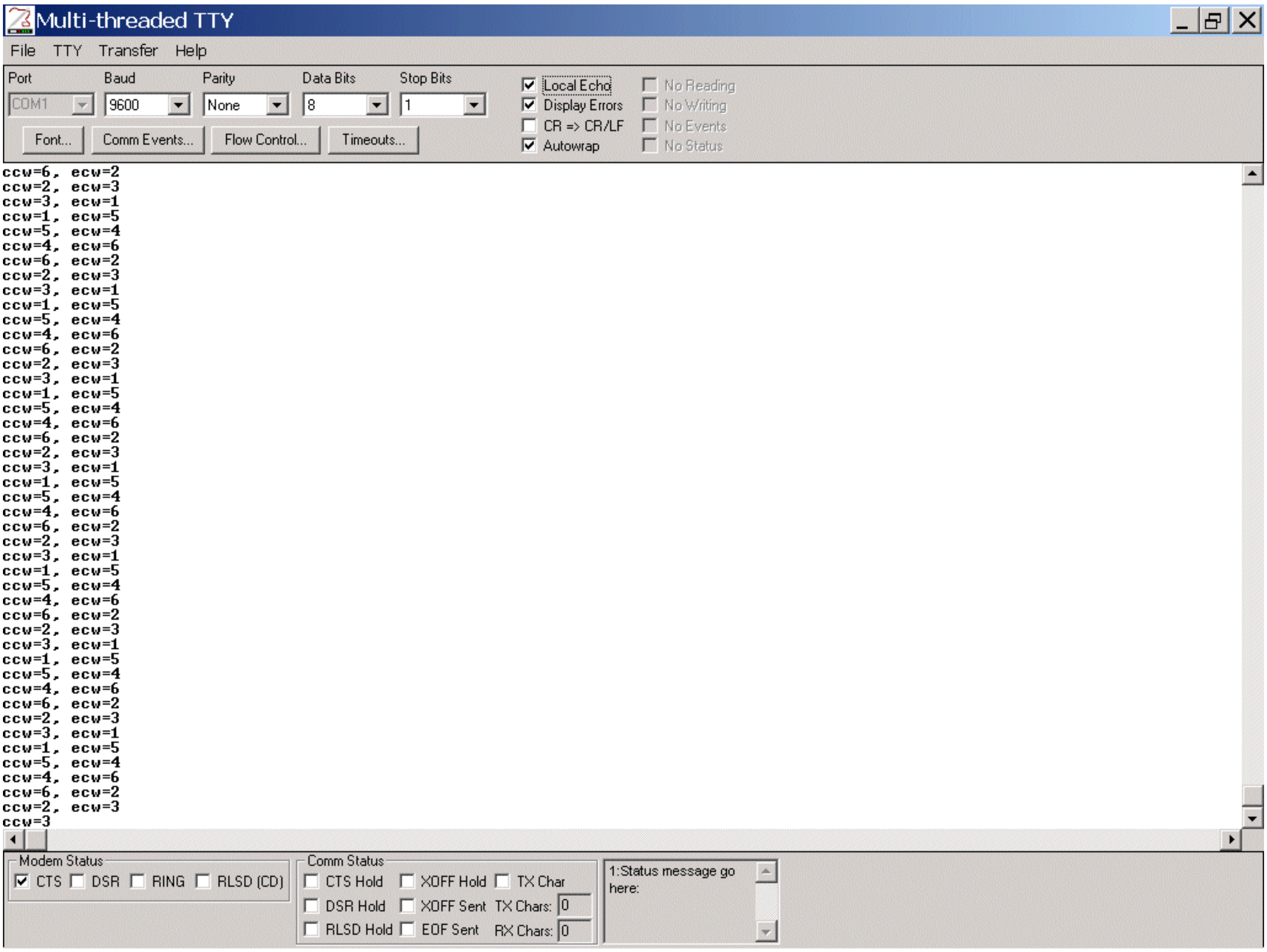

XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A66
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
351 //case '3': blinking=ON, printf(message3); break;
352 //}
353
354 // get hall state from P1H:
355 current_commutation_window = (unsigned char) P1H & 0x07;
356
357 if (current_commutation_window == 2)
358 printf("ccw=2, ");
359 if (current_commutation_window == 3)
360 printf("ccw=3, ");
361 if (current_commutation_window == 1)
362 printf("ccw=1, ");
363 if (current_commutation_window == 5)
364 printf("ccw=5, ");
365 if (current_commutation_window == 4)
366 printf("ccw=4, ");
367 if (current_commutation_window == 6)
368 printf("ccw=6, ");
369
370 printf("ecw=%u\r\n",((HallPatt[current_commutation_window]<<8)>>8)&0x07);
371
372 // program shadow registers with next hall state:
373 CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;
374
375 // program shadow registers with next output pattern:
376 CCU6_MCMOUTS |= OutputPatt[current_commutation_window];
377
378 // for this programming example we will do the shadow transfer always manually:
379 CCU6_MCMOUTS |= 0x8080;
380 }
381 // USER CODE END
382
383 } // End of function main
384
MAIN.C
Build Command Find in Files
ULINK Driver for XC16x
L:357 C:5
R/W

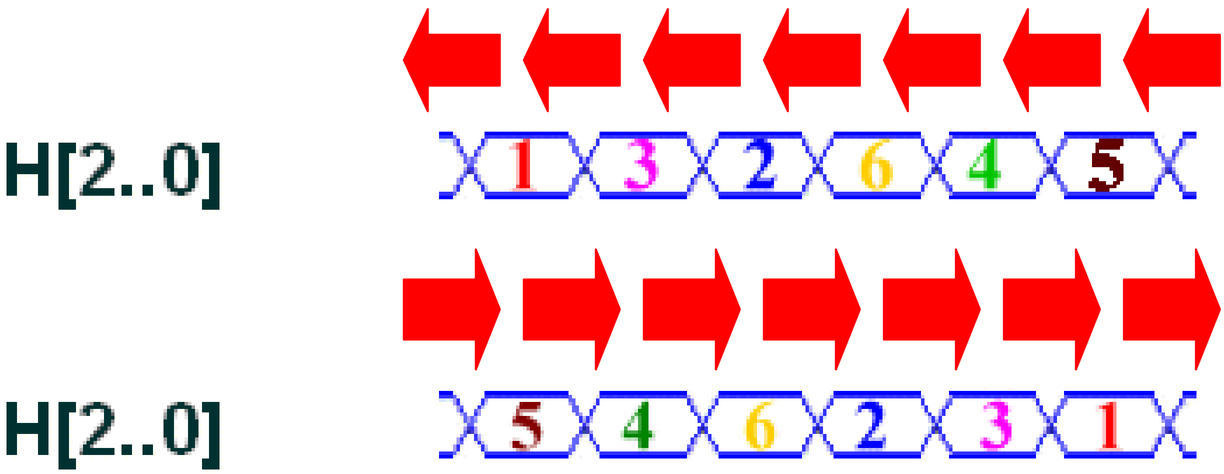
Build Application:

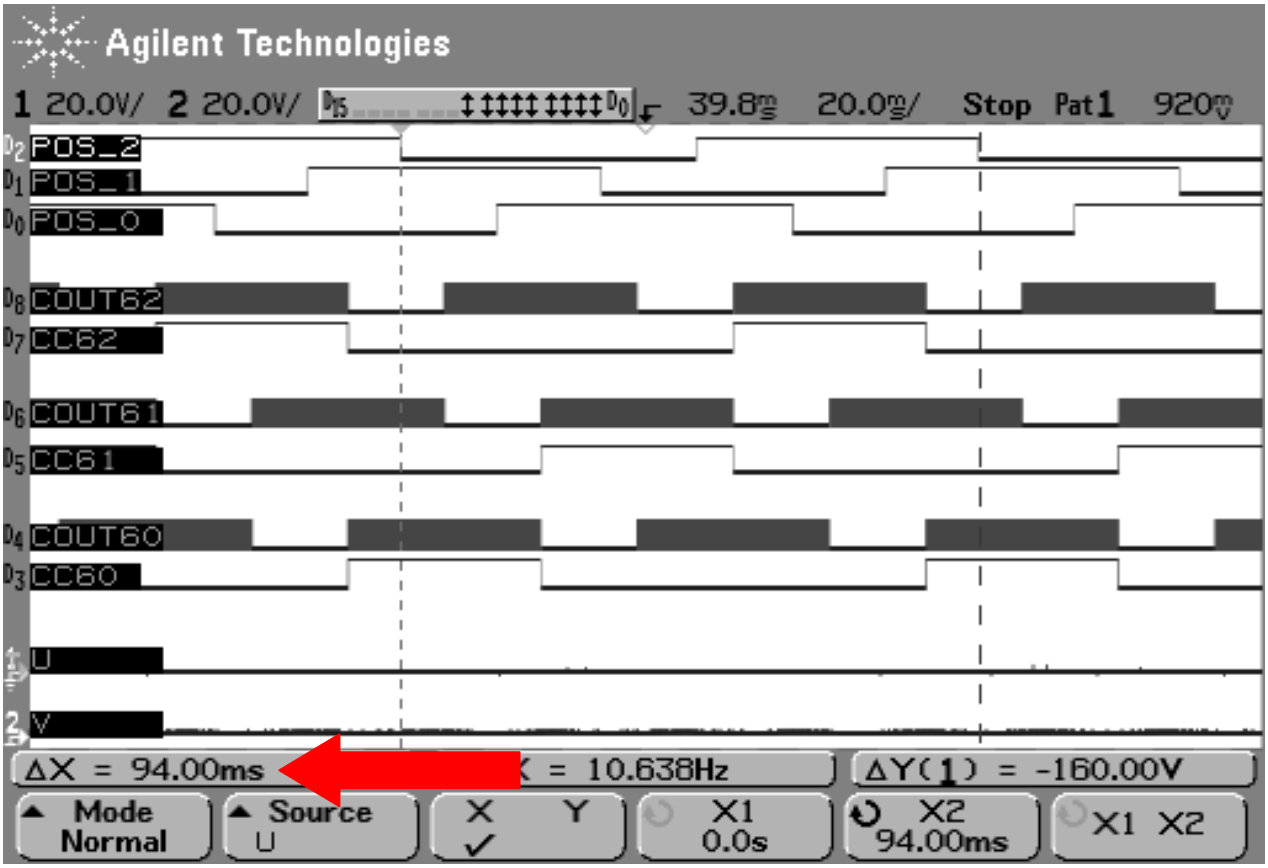


The Motor is running ! ☺

See the result:





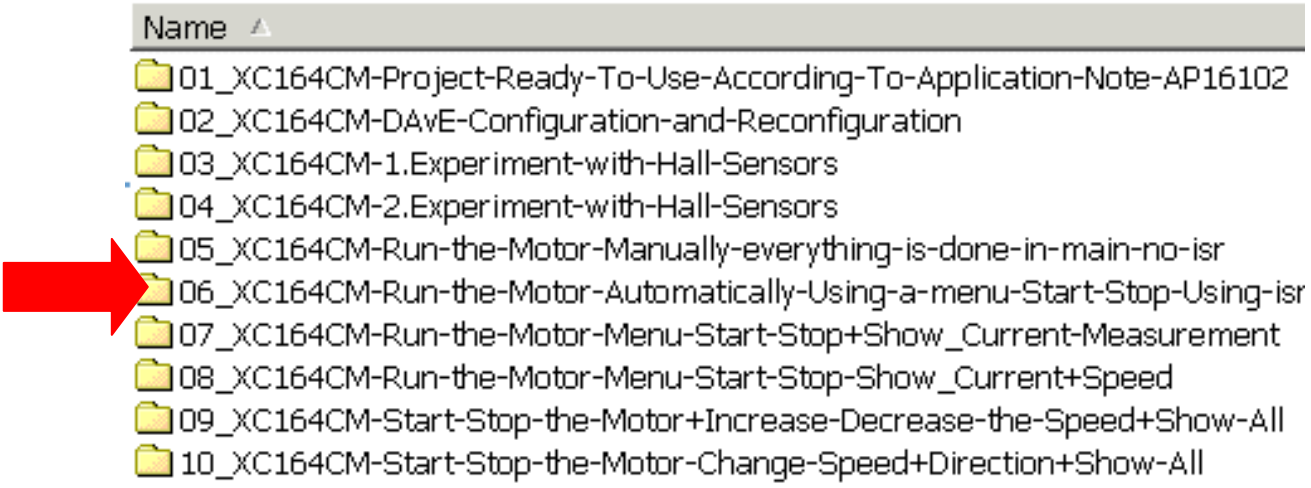


revolutions per minute [rpm]	revolutions per second [rps]	time for 1 revolution [ms]	time for 1 commutation window [ms]	commutation windows per second []
638	10,64	94	15,76	64
1000	16,67	60	10,00	100
2000	33,33	30	5,00	200
3000	50,00	20	3,33	300
4000	66,67	15	2,50	400
5000	83,33	12	2,00	500
6000	100,00	10	1,67	600

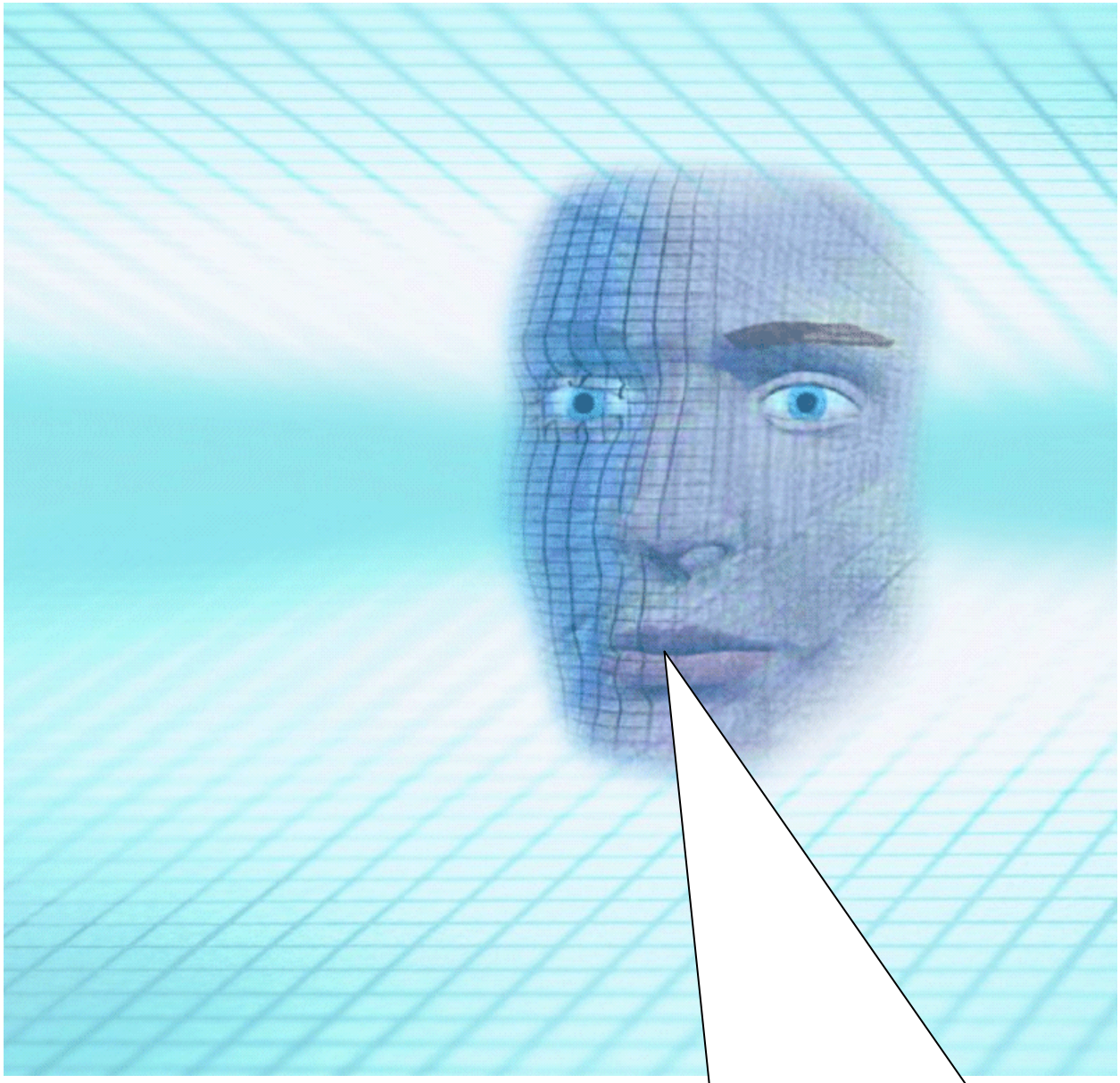
6.)

Running the Motor Automatically Using a Menu [Using a CCU6-interrupt-function] :

Relevant Project:



Insert your application specific program:



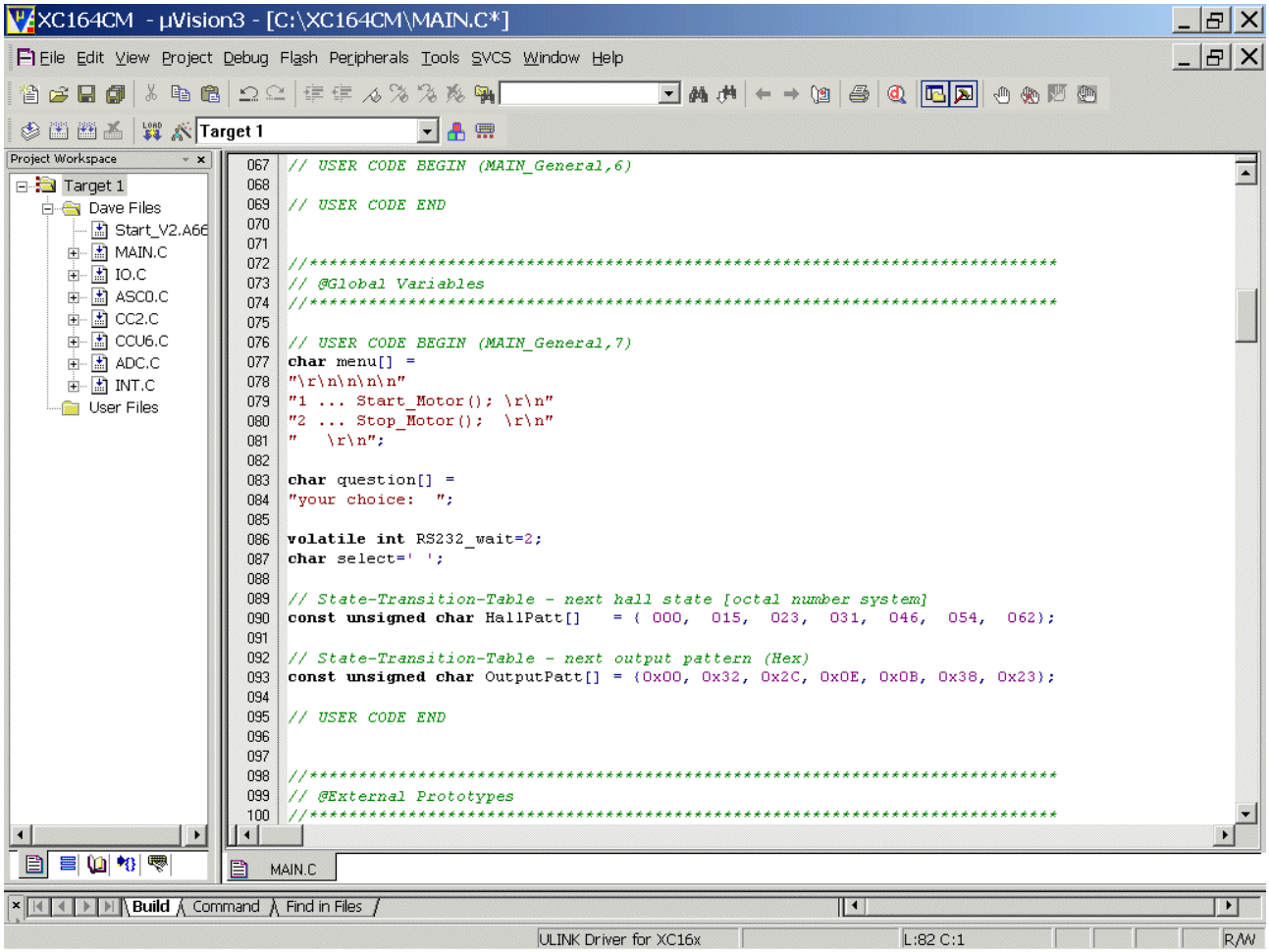
Note:
DAvE doesn’t change code which is inserted between ‘// USER CODE BEGIN’ and ‘// USER CODE END’. Therefore, whenever adding code to DAvE’s generated code, write it between ‘// USER CODE BEGIN’ and ‘// USER CODE END’.
If you wish to change DAvE´s generated code or add code outside these ‘USER CODE’ sections you will have to insert/modify your changes each time after letting DAvE regenerate code!

Double click MAIN.C and change Global Variable from

```
char menu[] =
"\r\n\n\n\n"
"1 ...            \r\n"
"2 ...            \r\n"
"   \r\n";
```

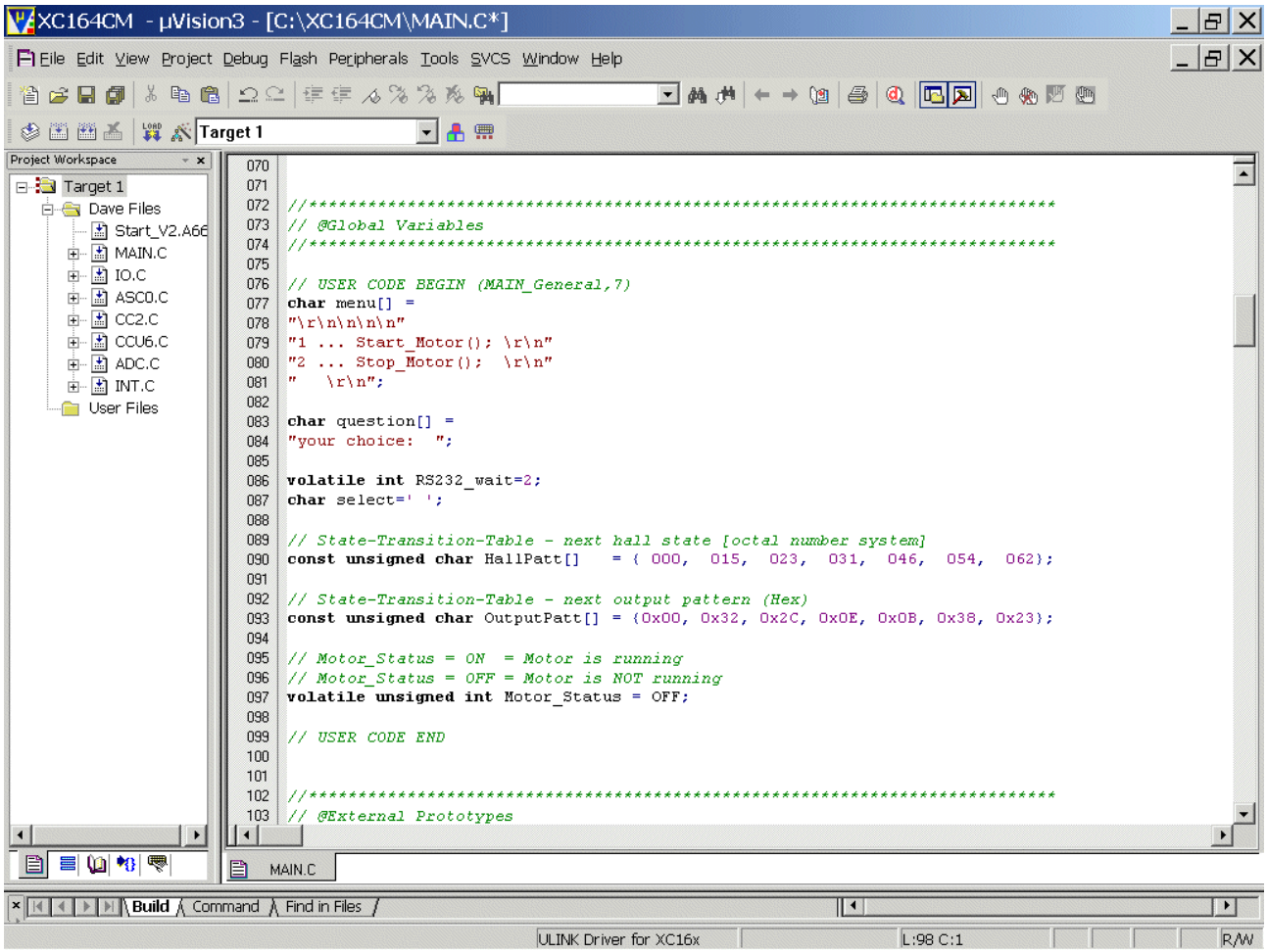
to:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor(); \r\n"
"2 ... Stop_Motor();  \r\n"
"   \r\n";
```



Double click MAIN.C and insert Global Variable:

```
// Motor_Status = ON  = Motor is running
// Motor_Status = OFF = Motor is NOT running
volatile unsigned int Motor_Status = OFF;
```

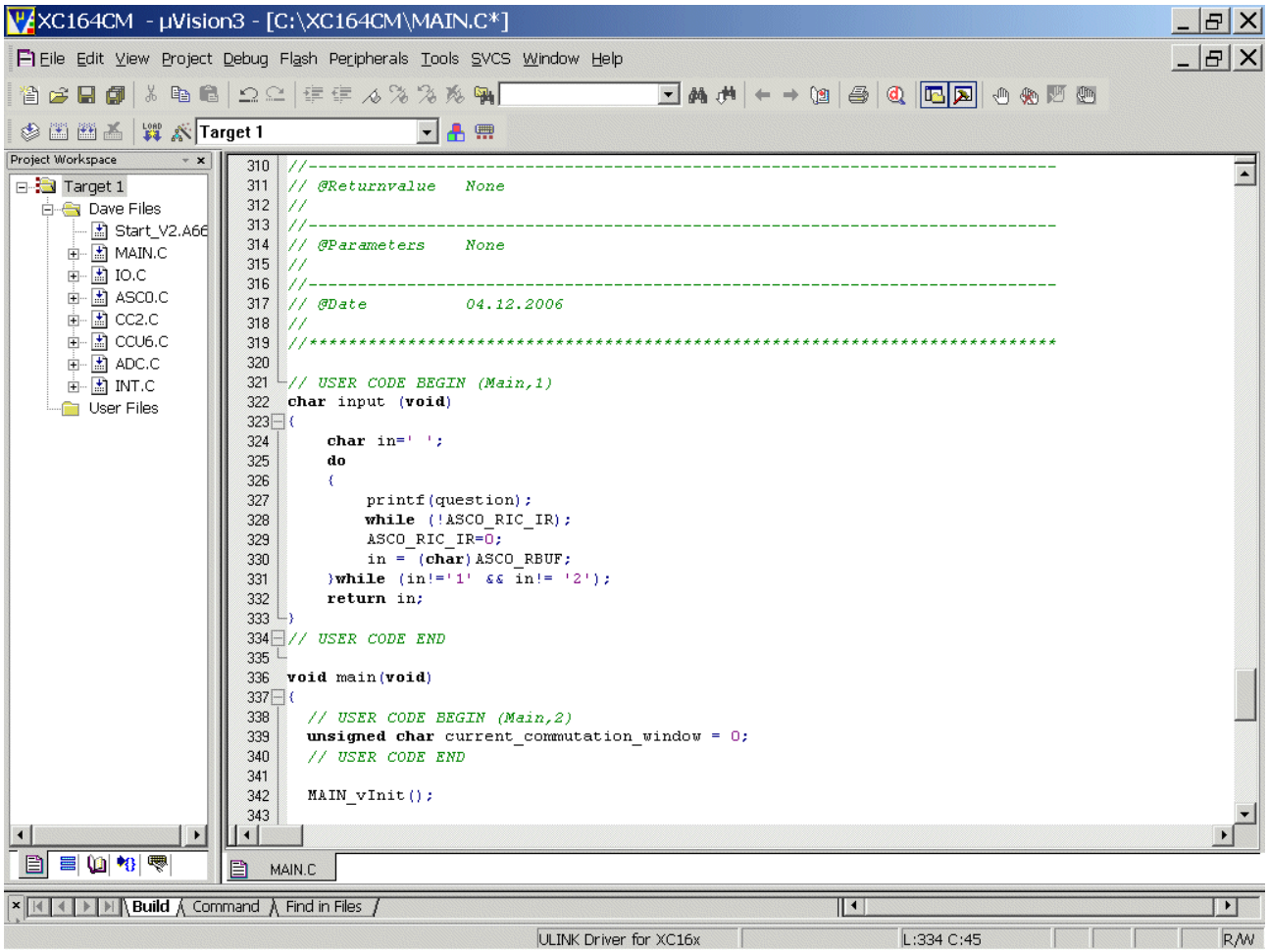


Double click MAIN.C and change Code from

```
char input (void)
{
     char in=' ';
     do
     {
          printf(question);
          while (!ASC0_RIC_IR);
          ASC0_RIC_IR=0;
          in = (char)ASC0_RBUF;
     }while (in!='1' && in!= '2' && in != '3');
     return in;
}
```

to:

```
char input (void)
{
     char in=' ';
     do
     {
          printf(question);
          while (!ASC0_RIC_IR);
          ASC0_RIC_IR=0;
          in = (char)ASC0_RBUF;
     }while (in!='1' && in!= '2');
     return in;
}
```

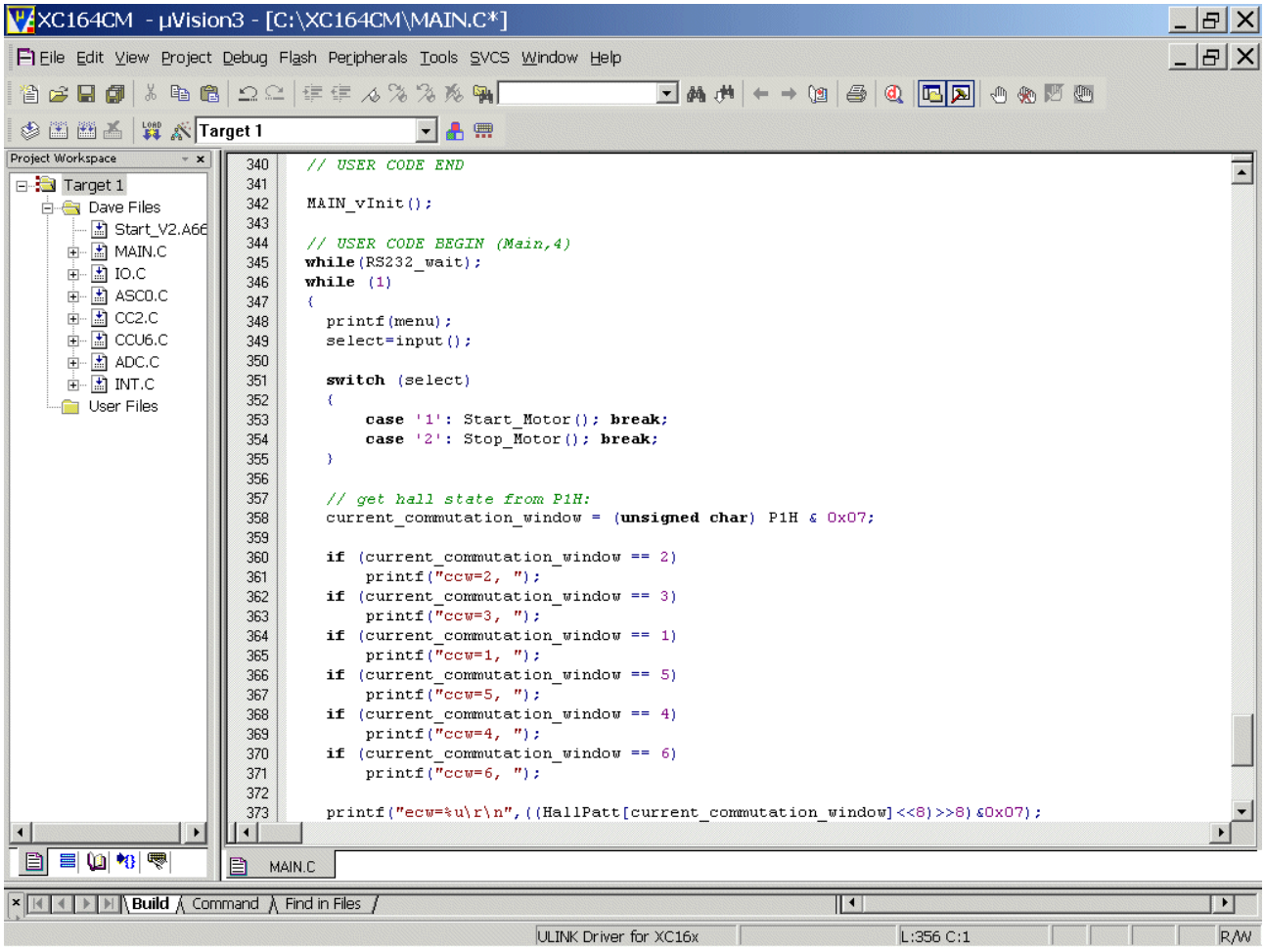

XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A66
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
310 //---
311 // @Returnvalue None
312 //
313 //---
314 // @Parameters None
315 //
316 //---
317 // @Date 04.12.2006
318 //
319 //***
320
321 // USER CODE BEGIN (Main,1)
322 char input (void)
323 {
324 char in=' ';
325 do
326 {
327 printf(question);
328 while (!ASC0_RIC_IR);
329 ASC0_RIC_IR=0;
330 in = (char)ASC0_RBUF;
331 }while (in!='1' && in!= '2');
332 return in;
333 }
334 // USER CODE END
335
336 void main(void)
337 {
338 // USER CODE BEGIN (Main,2)
339 unsigned char current_commutation_window = 0;
340 // USER CODE END
341
342 MAIN_vInit();
343
MAIN.C
Build Command Find in Files
ULINK Driver for XC16x
L:334 C:45
R/W

Double click MAIN.C and change from

```
//printf(menu);
//select=input();

//switch (select)
//{
     //case '1': ; break;
     //case '2': ; break;
//}
```

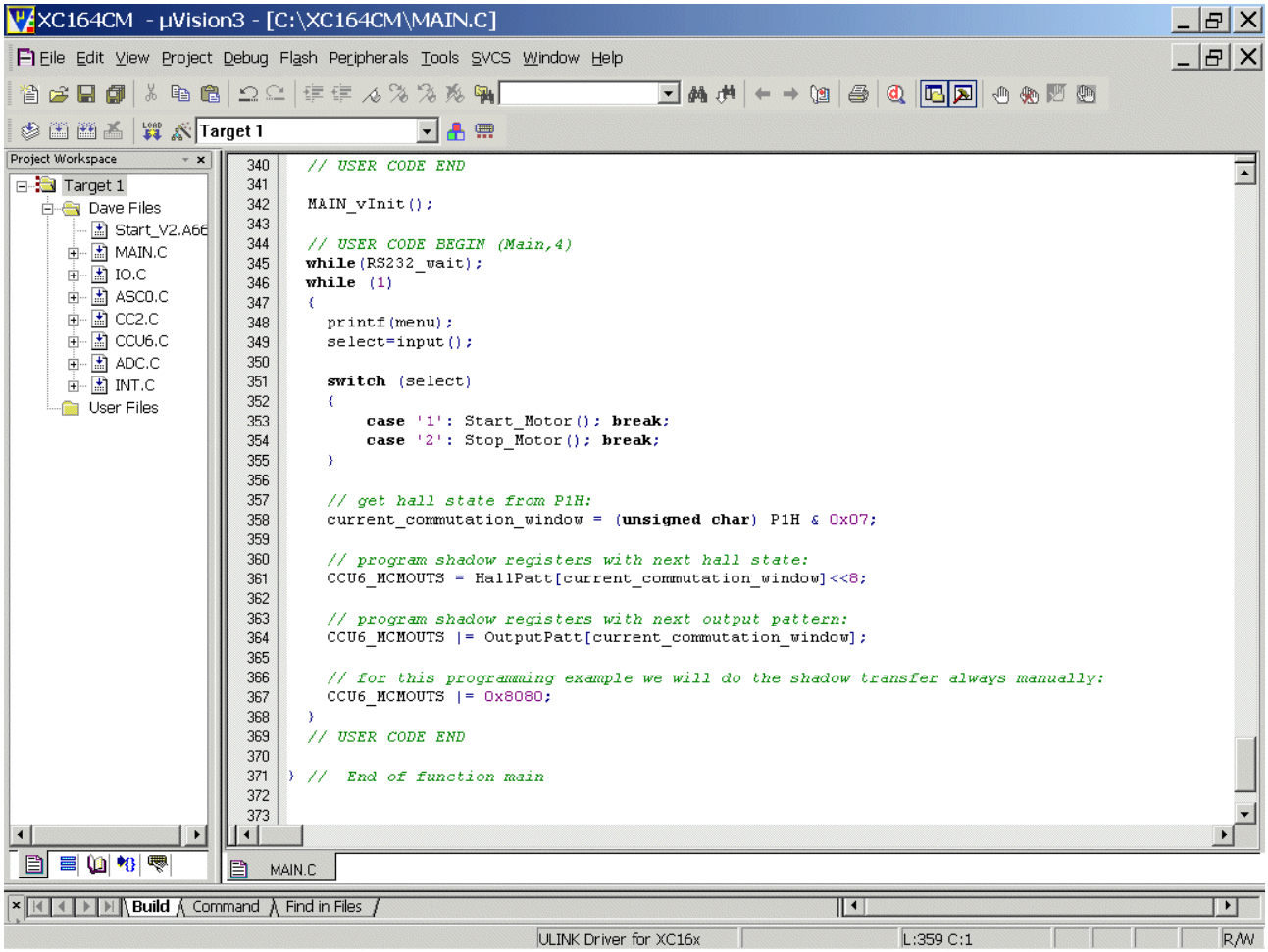
to:

```
printf(menu);
select=input();

switch (select)
{
     case '1': Start_Motor(); break;
     case '2': Stop_Motor(); break;
}
```



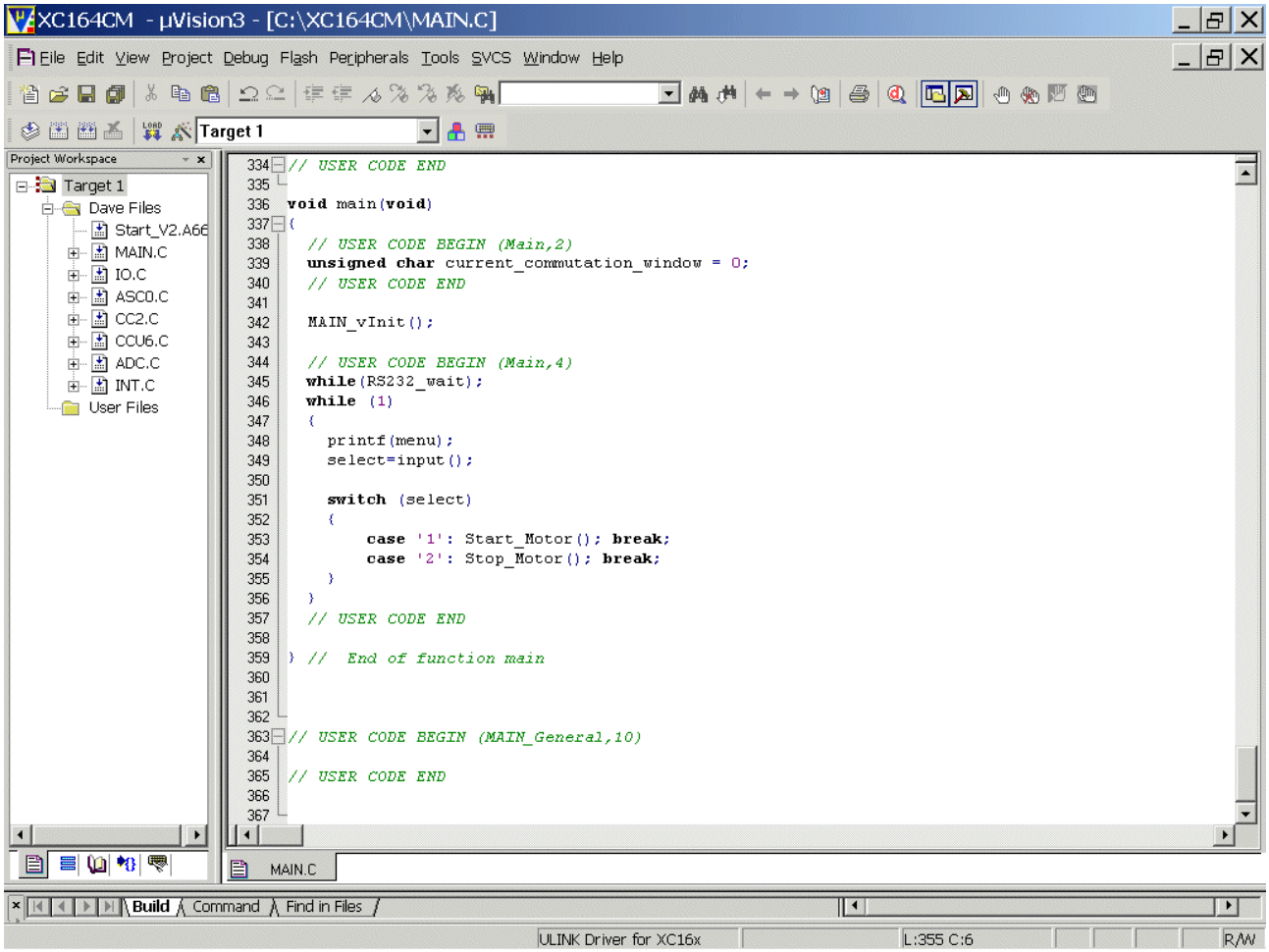
Double click MAIN.C and DELETE Code:

```
if (current_commutation_window == 2)
   printf("ccw=2, ");
if (current_commutation_window == 3)
   printf("ccw=3, ");
if (current_commutation_window == 1)
   printf("ccw=1, ");
if (current_commutation_window == 5)
   printf("ccw=5, ");
if (current_commutation_window == 4)
   printf("ccw=4, ");
if (current_commutation_window == 6)
   printf("ccw=6, ");

printf("ecw=%u\r\n",((HallPatt[current_commutation_window]<<8)>>8)&0x07);
```



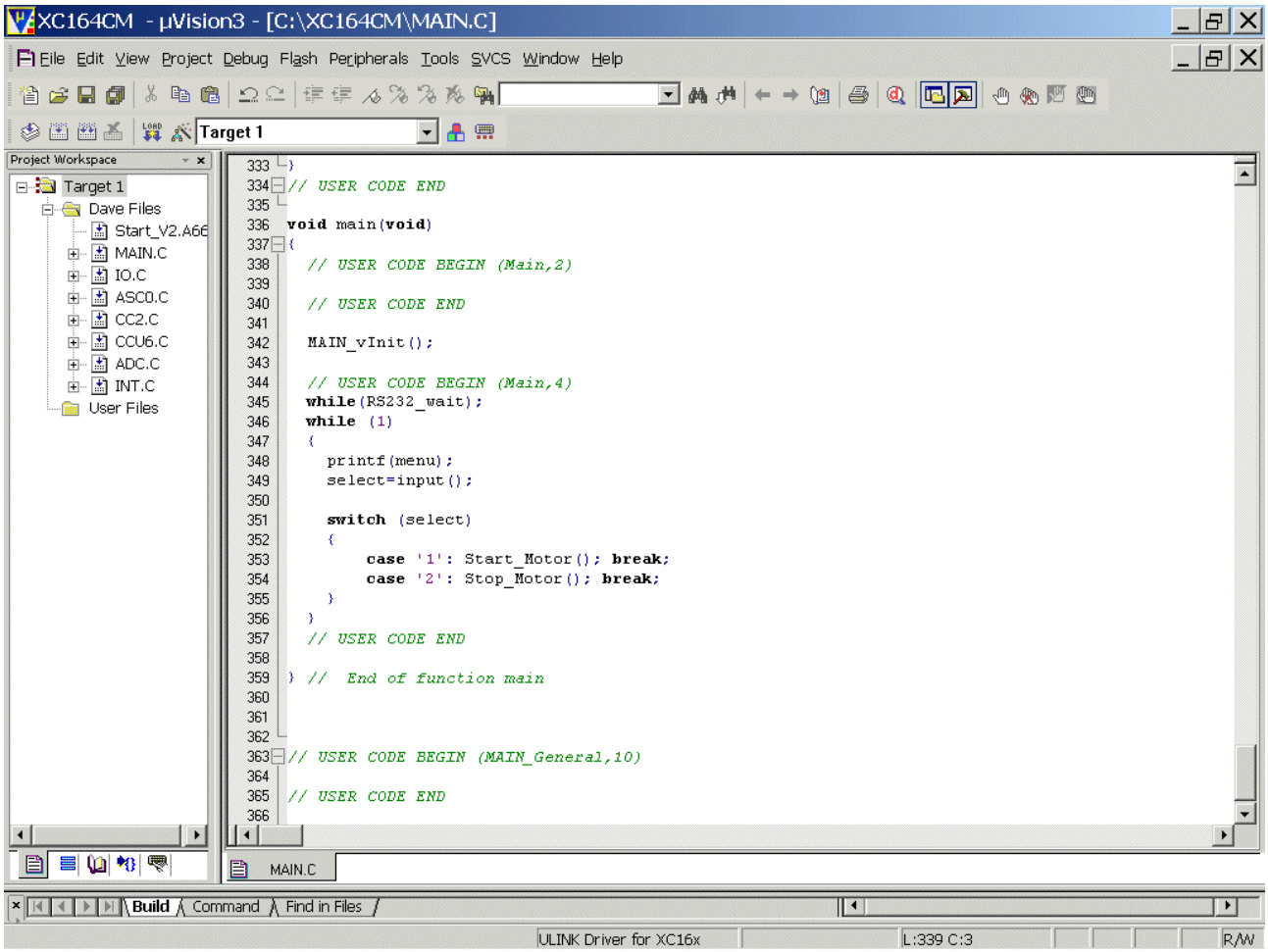
Double click MAIN.C and DELETE Code:

```
// get hall state from P1H:
current_commutation_window = (unsigned char) P1H & 0x07;

// program shadow registers with next hall state:
CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;

// program shadow registers with next output pattern:
CCU6_MCMOUTS |= OutputPatt[current_commutation_window];

// for this programming example we will do the shadow transfer always manually:
CCU6_MCMOUTS |= 0x8080;
```



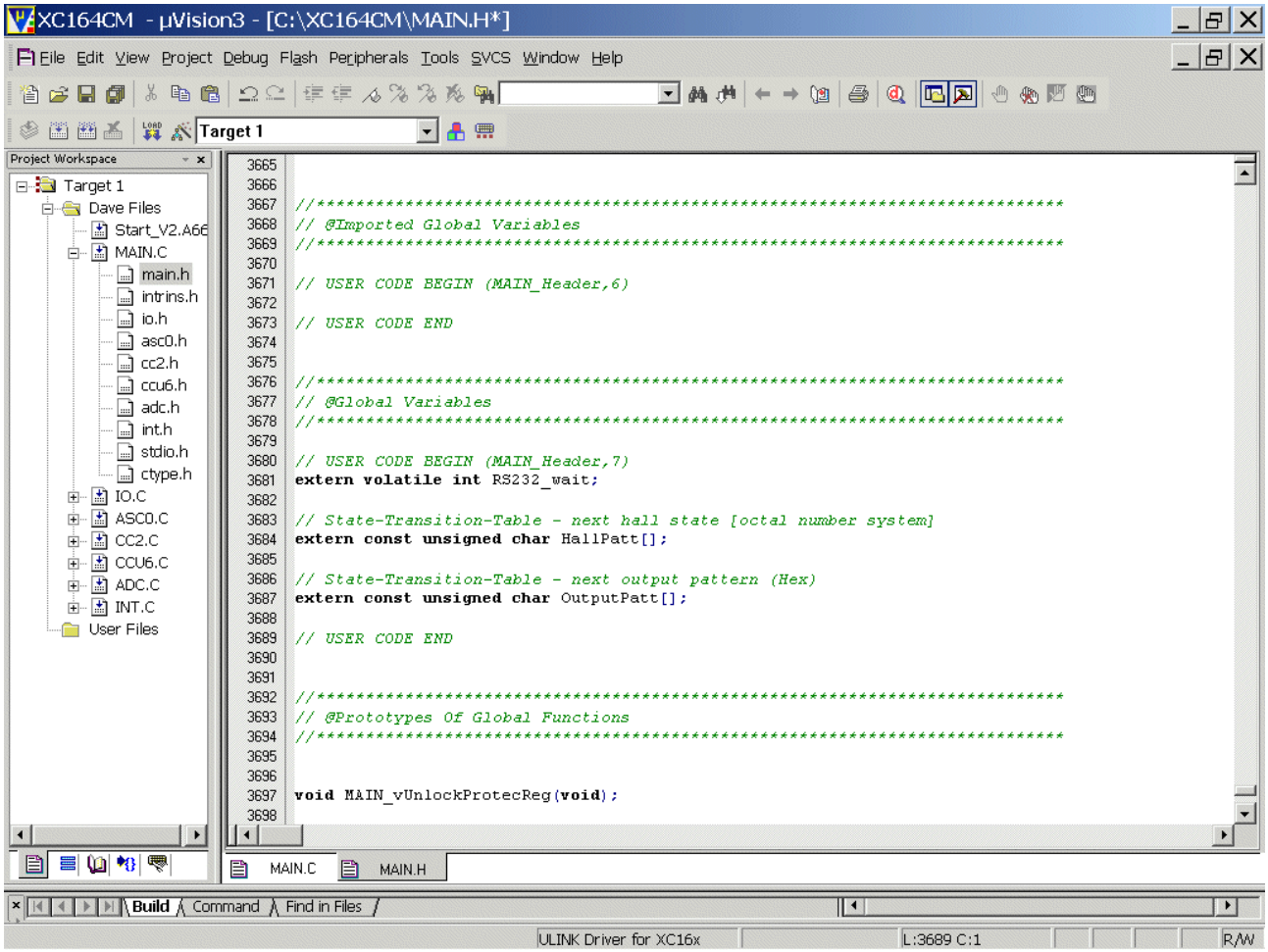
Double click MAIN.C and DELETE Code:

```
unsigned char current_commutation_window = 0;
```



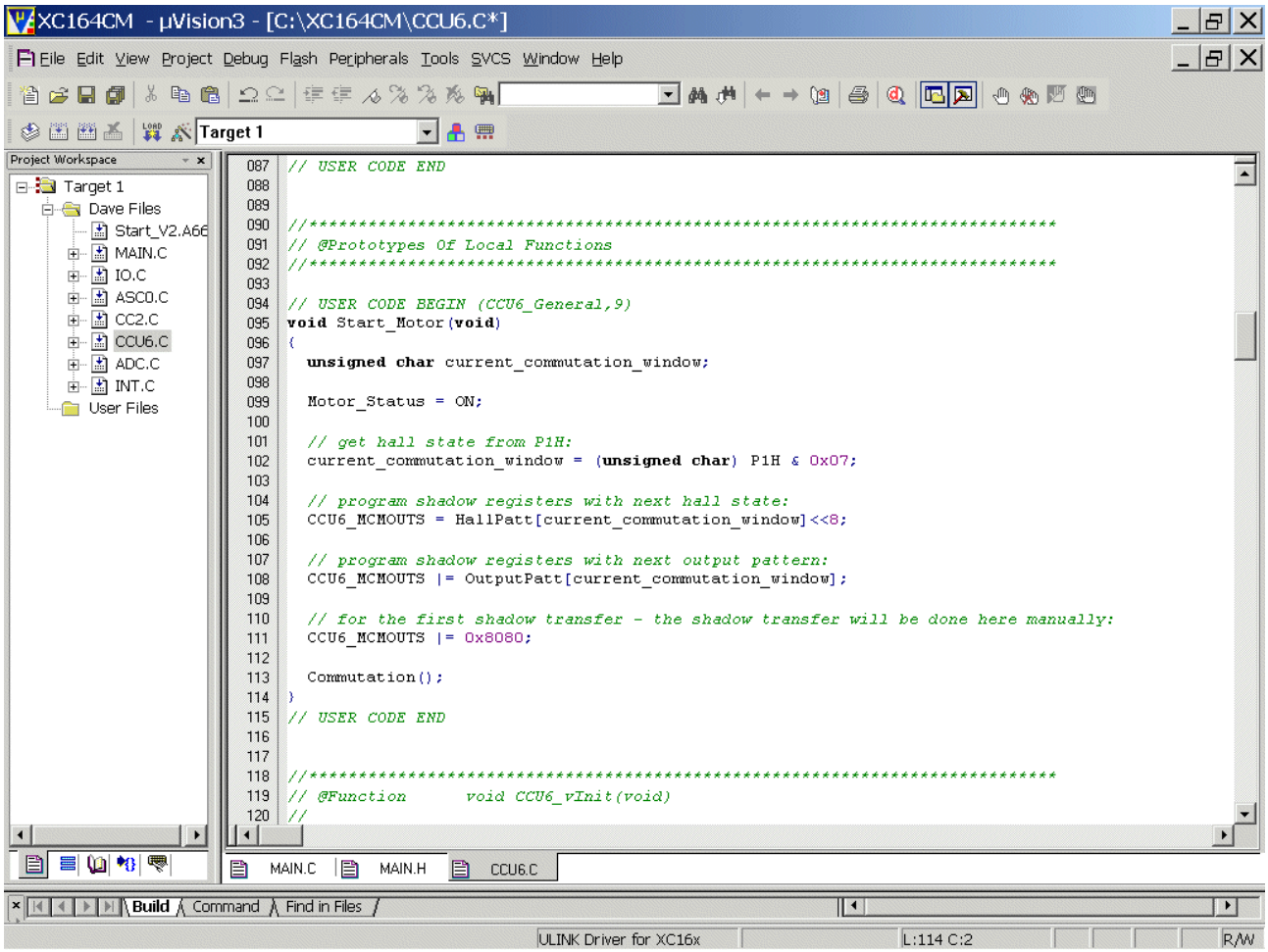
Double click main.h and insert Extern Declaration of Global Variables:

```
// State-Transition-Table - next hall state [octal number system]
extern const unsigned char HallPatt[];

// State-Transition-Table - next output pattern (Hex)
extern const unsigned char OutputPatt[];
```



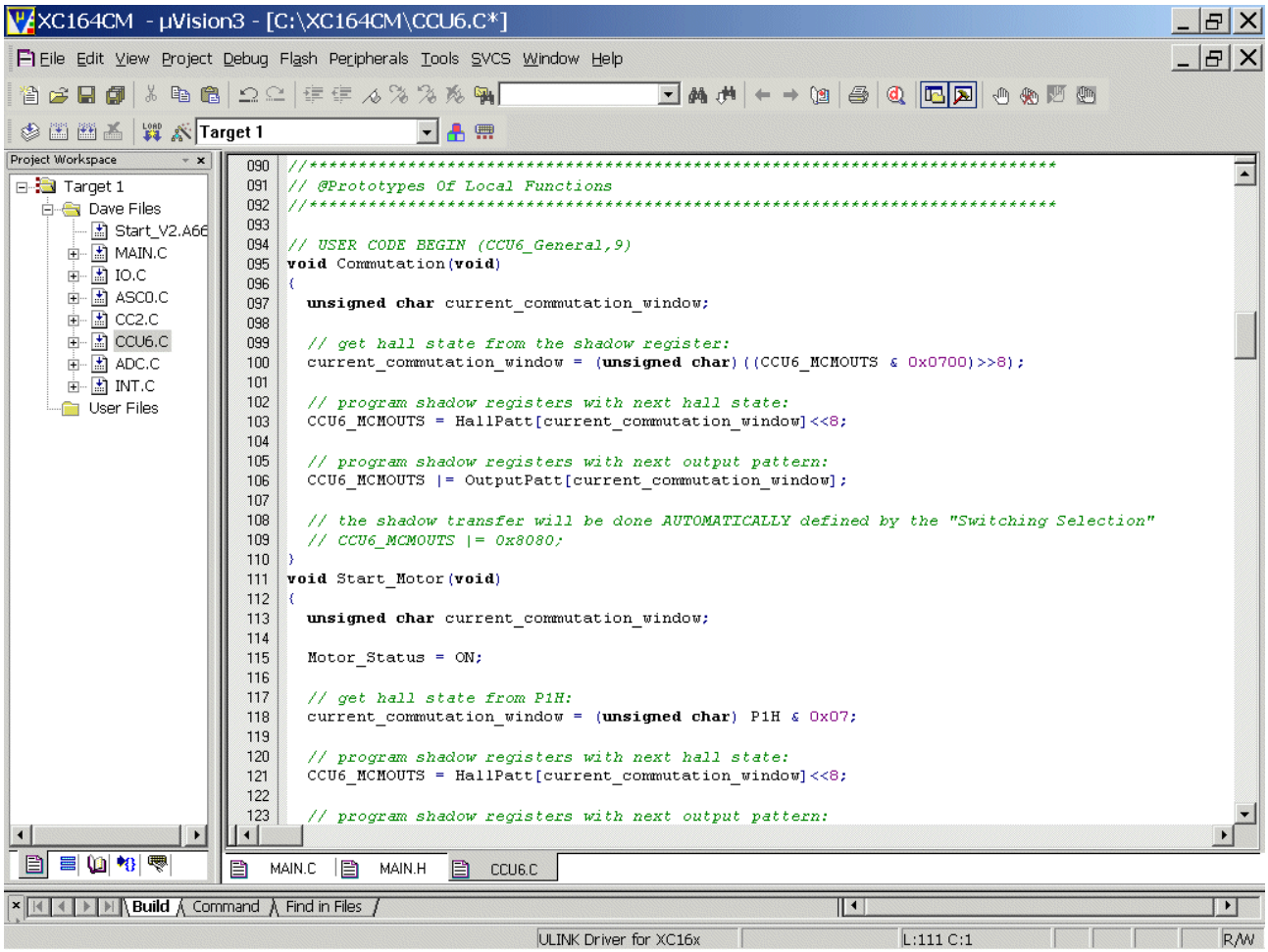
Double click CCU6.c and insert the following code:

```
void Start_Motor(void)
{
  unsigned char current_commutation_window;

  Motor_Status = ON;

  // get hall state from P1H:
  current_commutation_window = (unsigned char) P1H & 0x07;

  // program shadow registers with next hall state:
  CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;

  // program shadow registers with next output pattern:
  CCU6_MCMOUTS |= OutputPatt[current_commutation_window];

  // for the first shadow transfer - the shadow transfer will be
done here manually:
  CCU6_MCMOUTS |= 0x8080;

  Commutation();
}
```



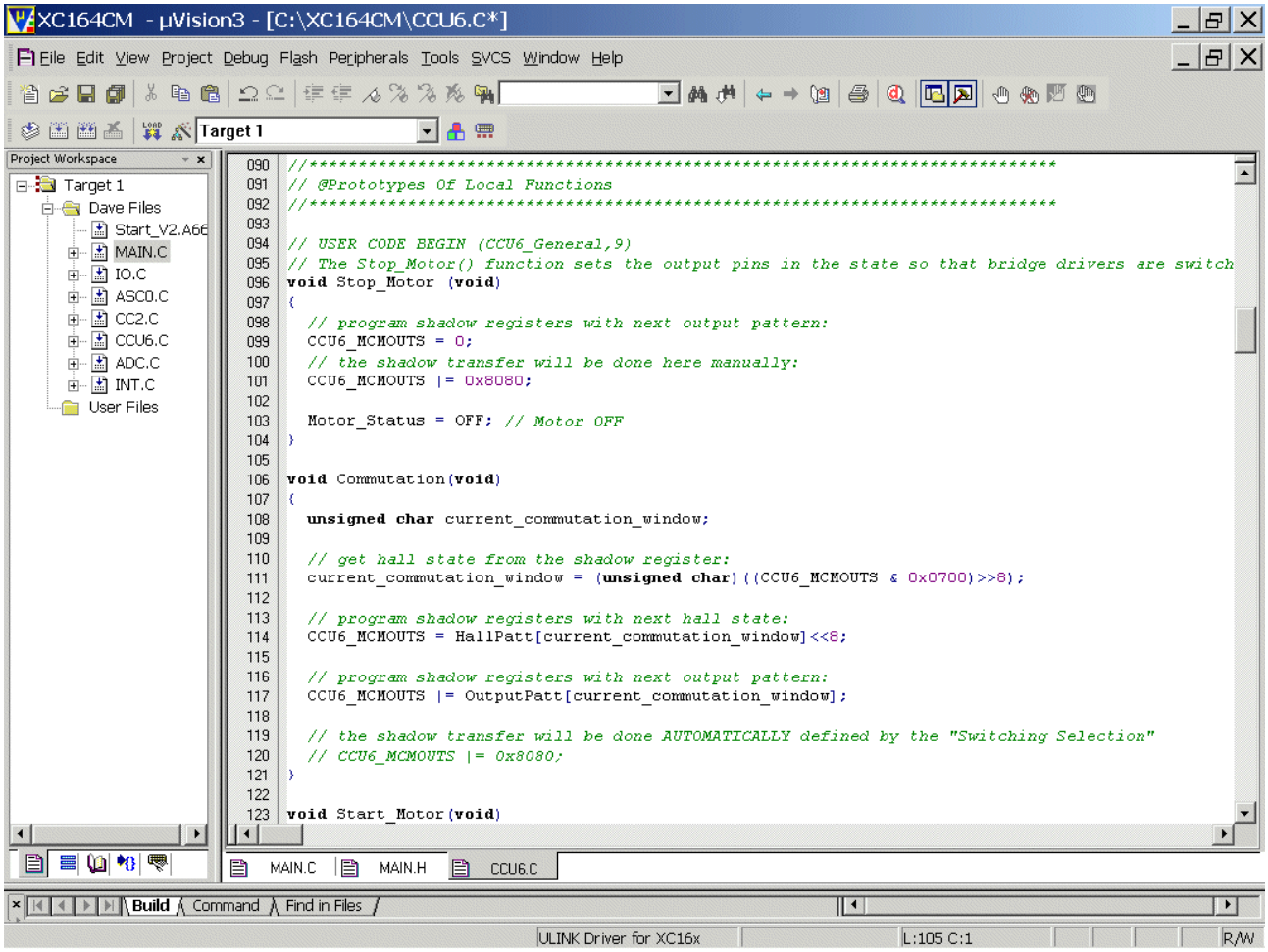
Double click CCU6.c and insert the following code [in front of "void Start_Motor(void)"]:

```
void Commutation(void)
{
  unsigned char current_commutation_window;

  // get hall state from the shadow register:
  current_commutation_window = (unsigned char)((CCU6_MCMOUTS &
0x0700)>>8);

  // program shadow registers with next hall state:
  CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;

  // program shadow registers with next output pattern:
  CCU6_MCMOUTS |= OutputPatt[current_commutation_window];

  // the shadow transfer will be done AUTOMATICALLY defined by the
"Switching Selection"
  // CCU6_MCMOUTS |= 0x8080;
}
```



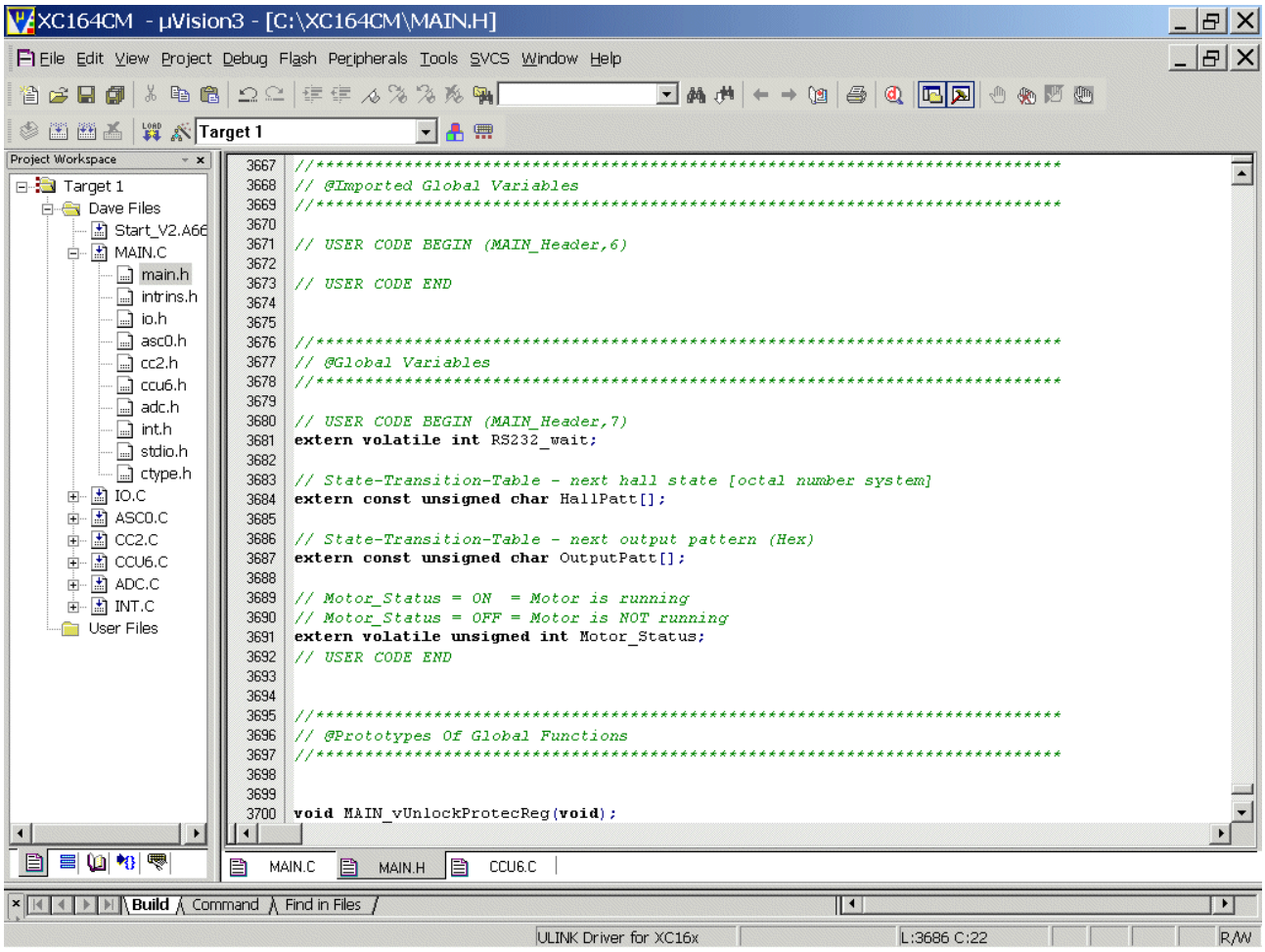
Double click CCU6.c and insert the following code [in front of "void Commutation(void)":

```
// The Stop_Motor() function sets the output pins in the state so
that bridge drivers are switched off
void Stop_Motor (void)
{
  // program shadow registers with next output pattern:
  CCU6_MCMOUTS = 0;
  // the shadow transfer will be done here manually:
  CCU6_MCMOUTS |= 0x8080;

  Motor_Status = OFF; // Motor OFF
}
```



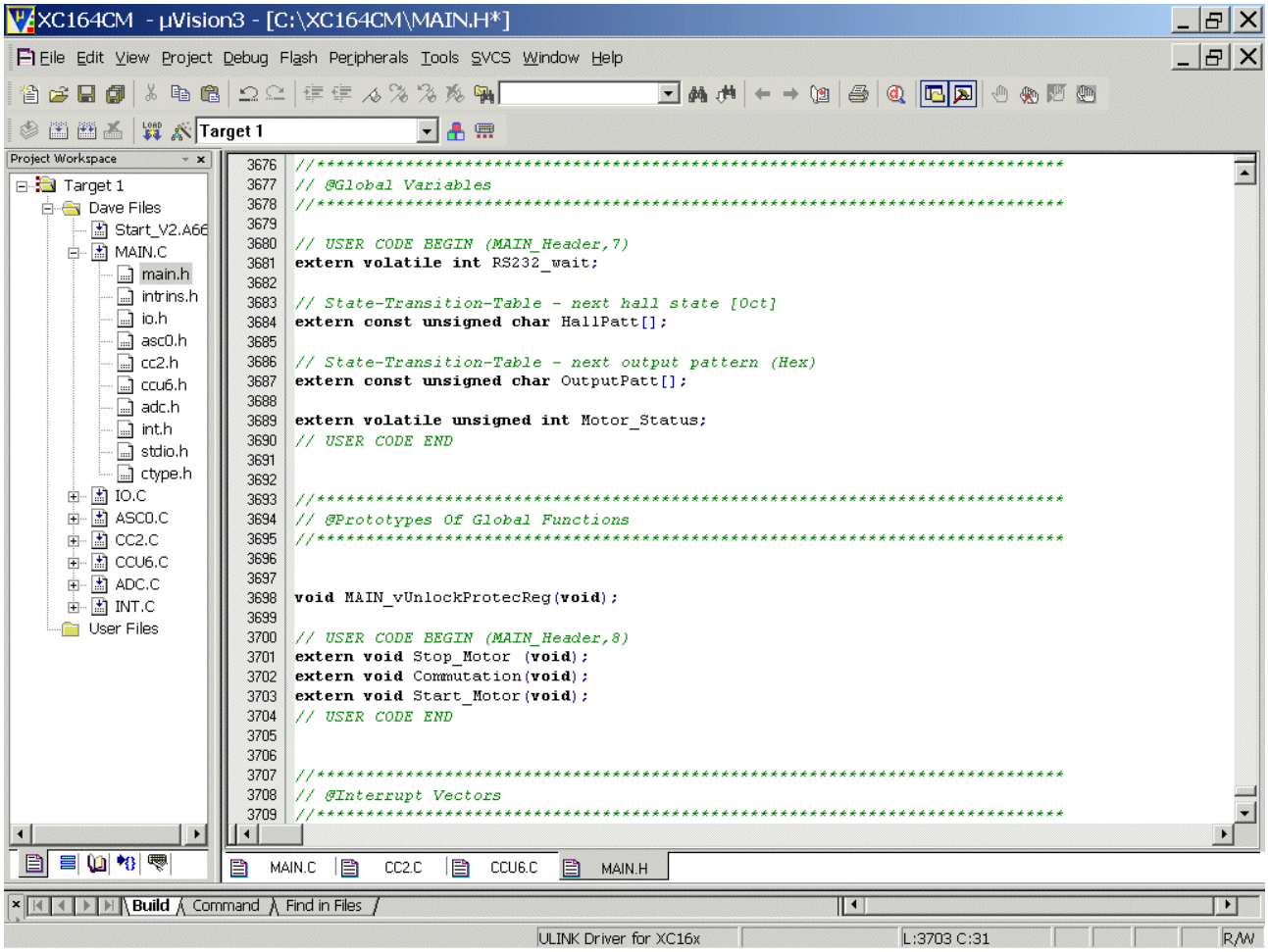
Double click main.h and insert Extern Declaration of Global Variable:

```
// Motor_Status = ON  = Motor is running
// Motor_Status = OFF = Motor is NOT running
extern volatile unsigned int Motor_Status;
```



Double click main.h and insert Prototypes of Global Functions:

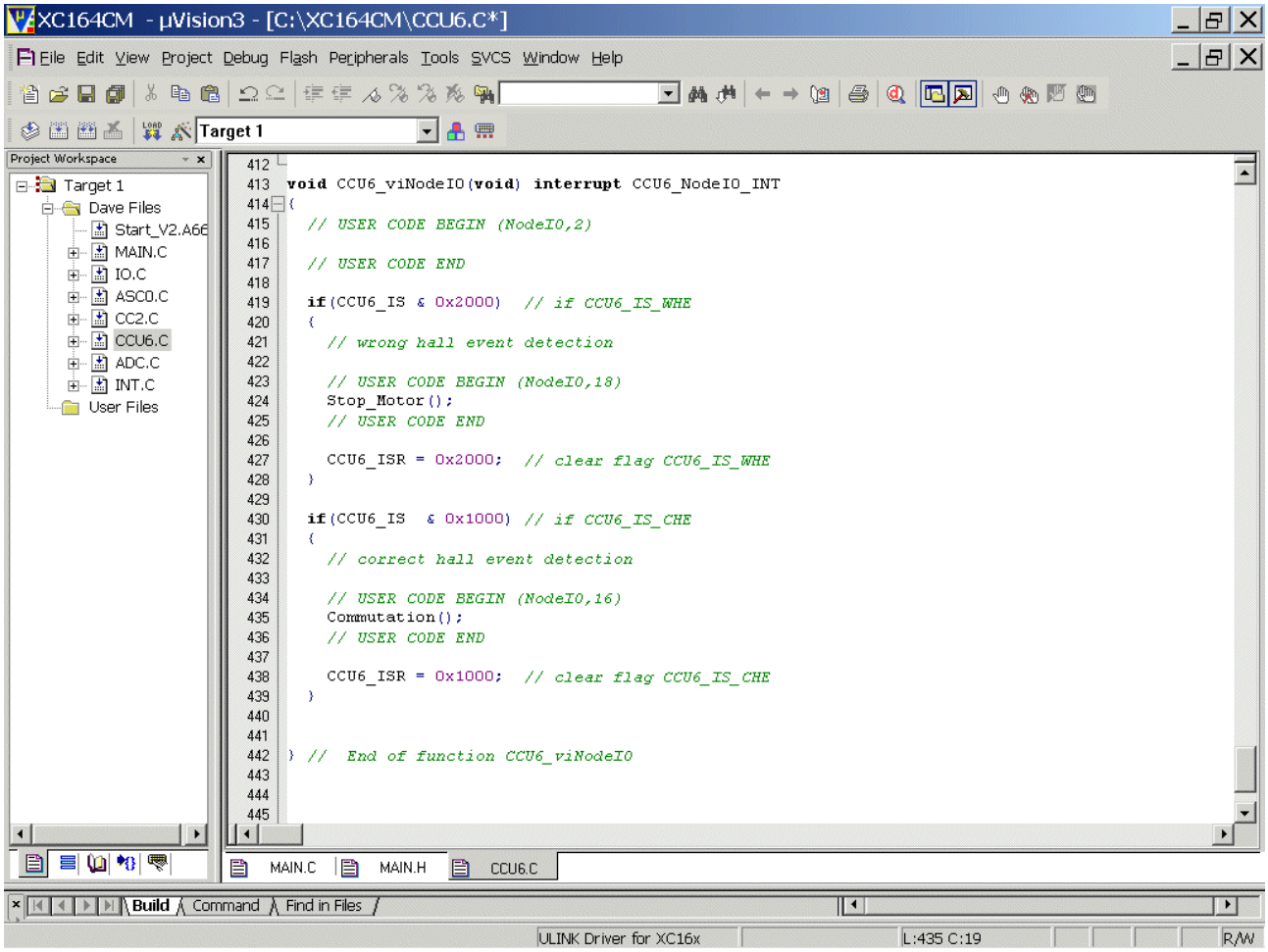
```
extern void Stop_Motor (void);
extern void Commutation(void);
extern void Start_Motor(void);
```

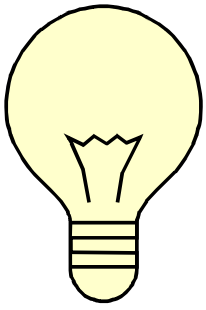


Double click CCU6.c and insert the following code:

```
Stop_Motor();
```

```
Commutation();
```



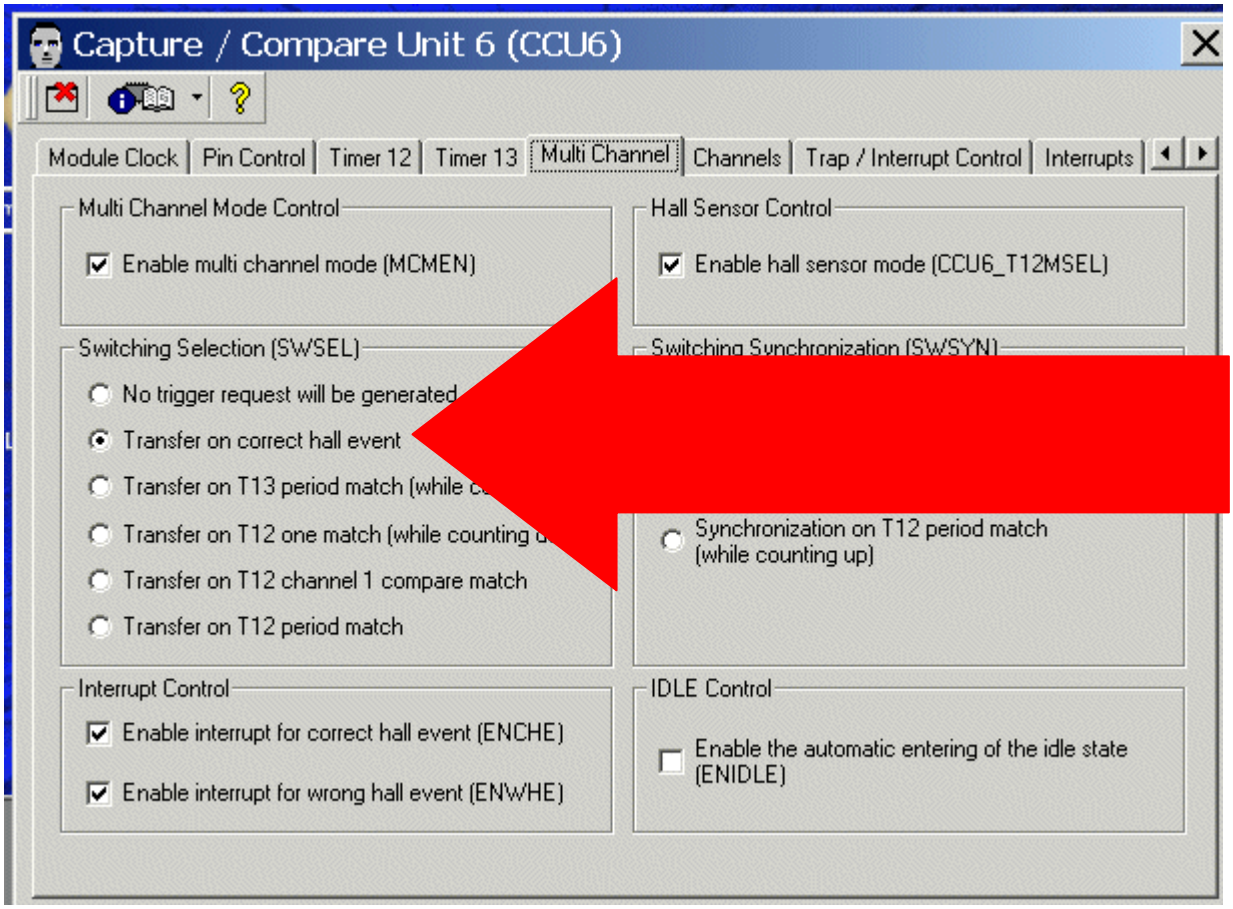


Note:

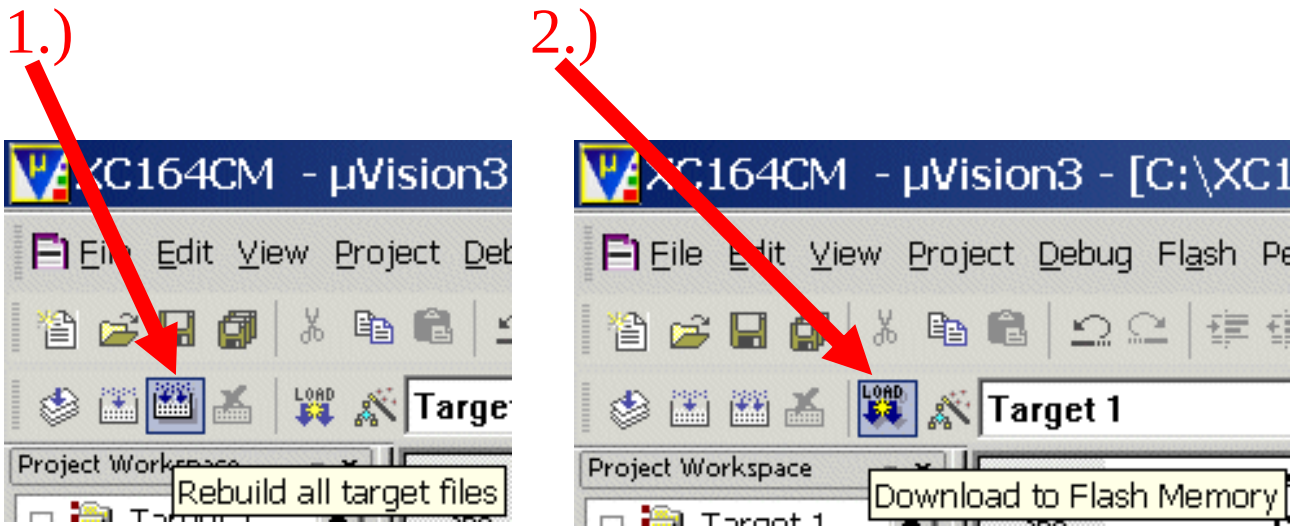
In case of a wrong Hall event, we will stop the motor.

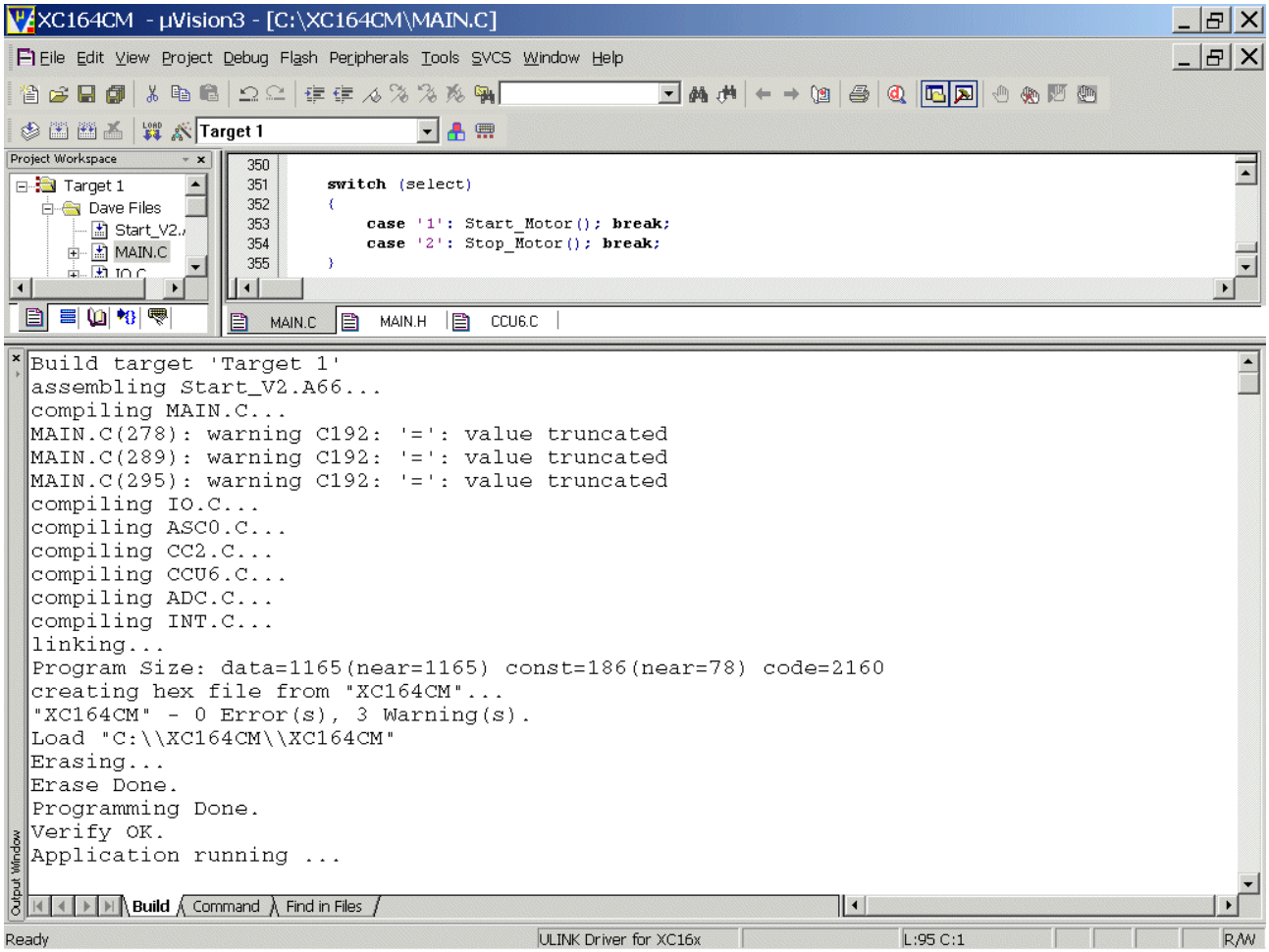
To update the PWM output pattern for every 60° commutation-window automatically, we are going to use the Correct-Hall-Event-Interrupt (to switch to the next commutation pattern).

Register Information / Additional Information:

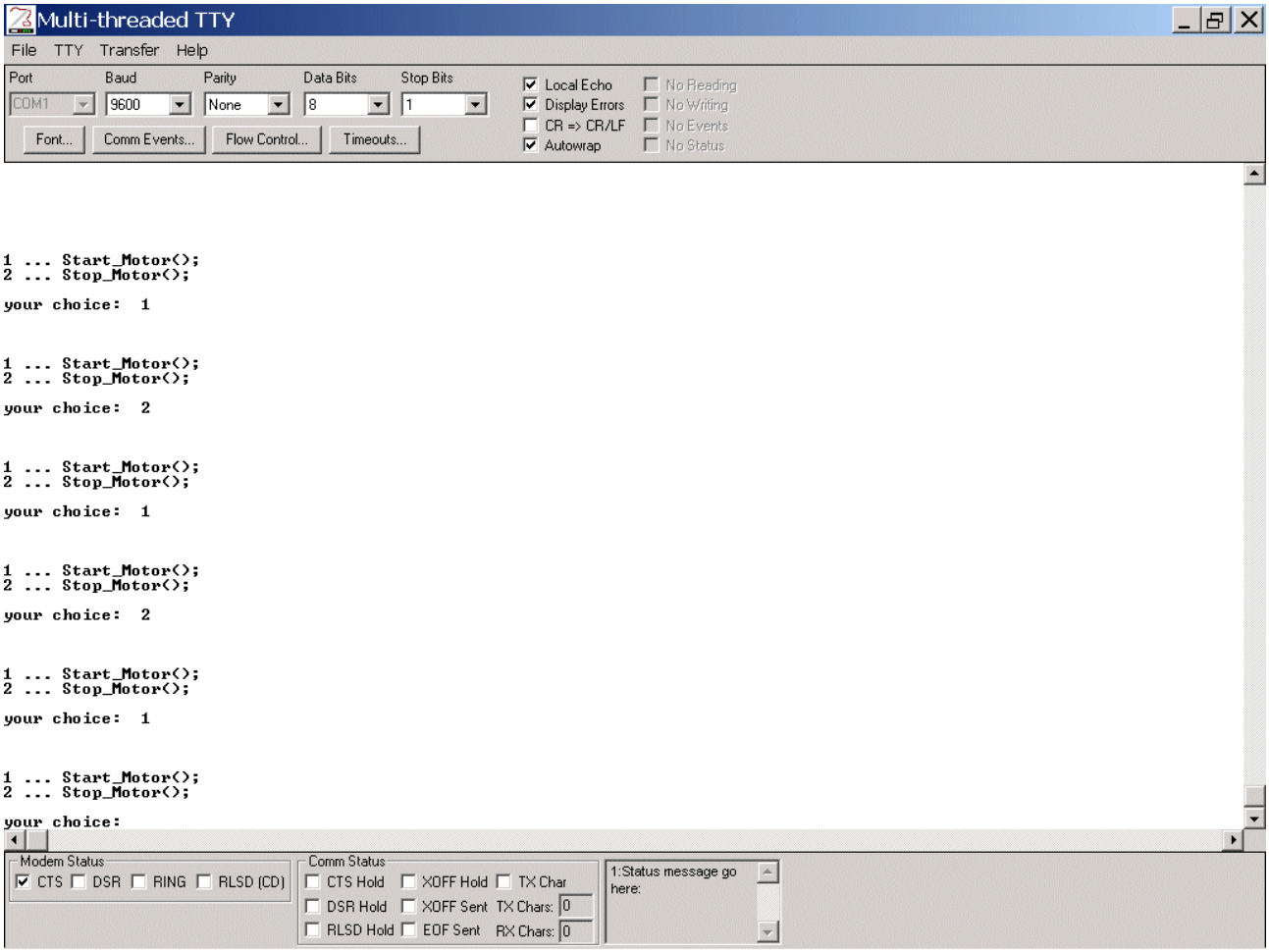


Build Application:





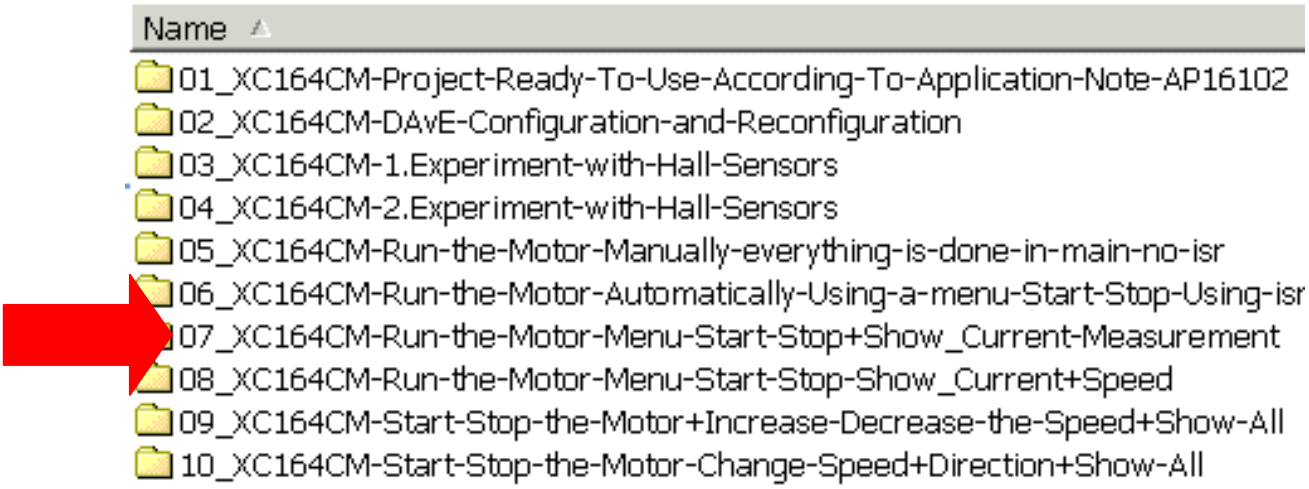
And see the result:

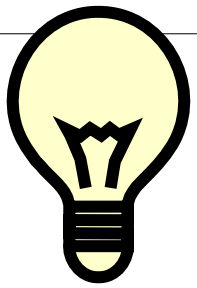


7.)

DC-Link Shunt Current Measurement:

Relevant Project:





Note:

Shunt:

In electronics,

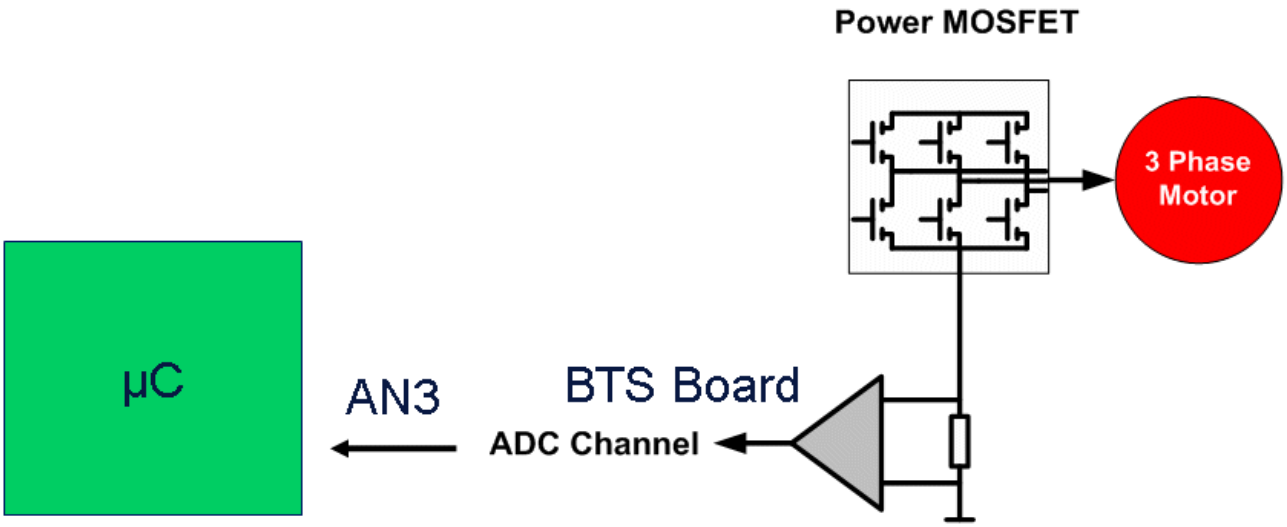
a shunt is a device which allows electrical current to pass around another point in the circuit.

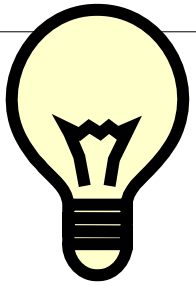
Use in current measuring:

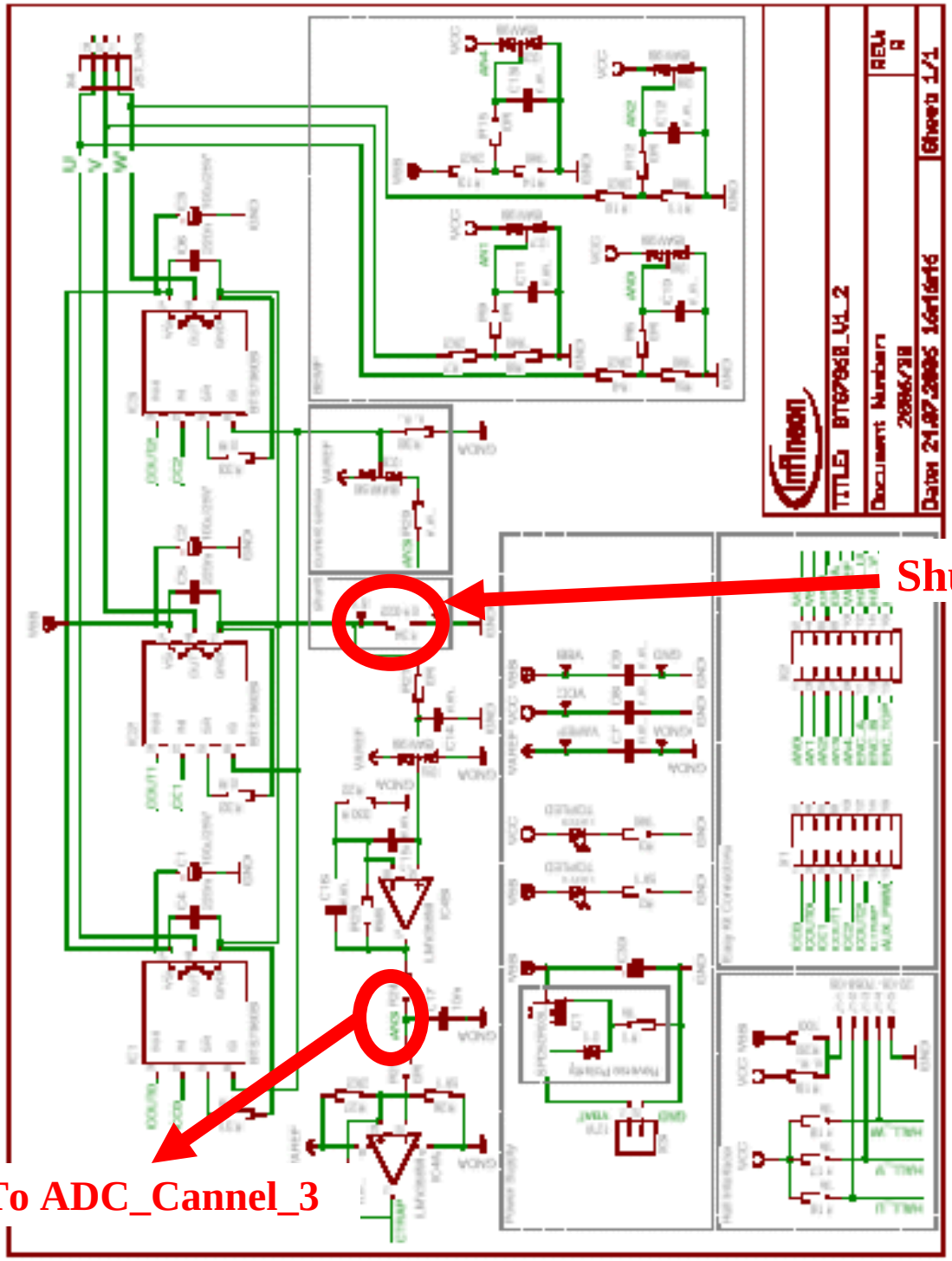
In this case a resistor of accurately-known resistance, the shunt, is placed in series so that all the current to be measured will flow through it. Since the resistance is known, by measuring the voltage drop across it, one can calculate the current flowing.

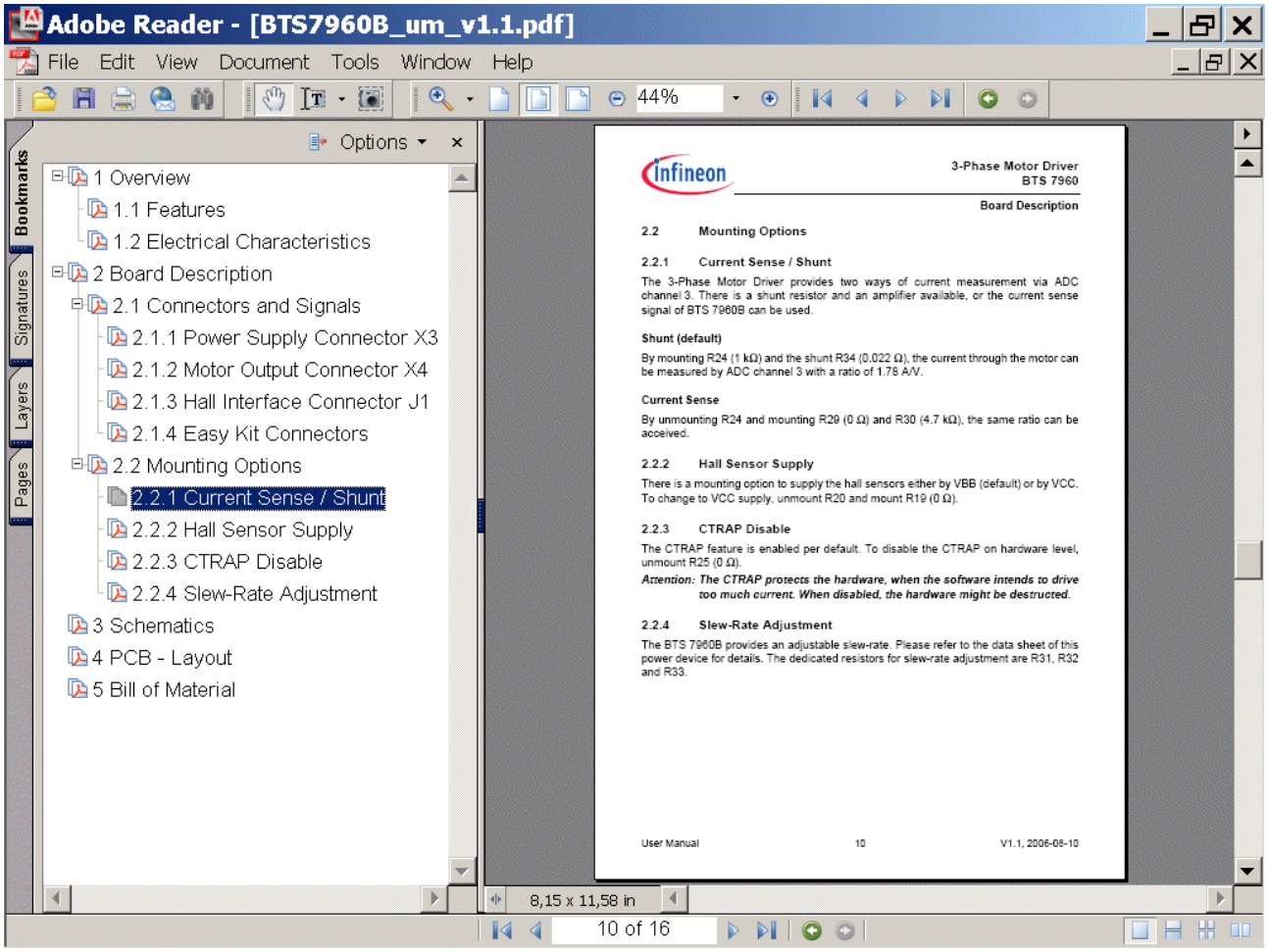
In order not to disrupt the circuit, the resistance of the shunt is normally very small. Shunts are rated by maximum current and voltage drop at that current, for example, a 500 A/50 mV shunt would have a maximum allowable current of 500 amps and at that current the voltage drop would be 50 millivolts. By convention, most shunts are designed to drop 50 mV when operating at their full rated current and most "ammeters" are actually designed as voltmeters that reach full-scale deflection at 50 mV.

If the current being measured is also at a high voltage potential this voltage will be present in the enclosure containing the reading instrument. Sometimes, the shunt is inserted in the return leg (low voltage side) to avoid this problem. Another solution is to use a Hall effect (non-contact) current sensor instead of a shunt.

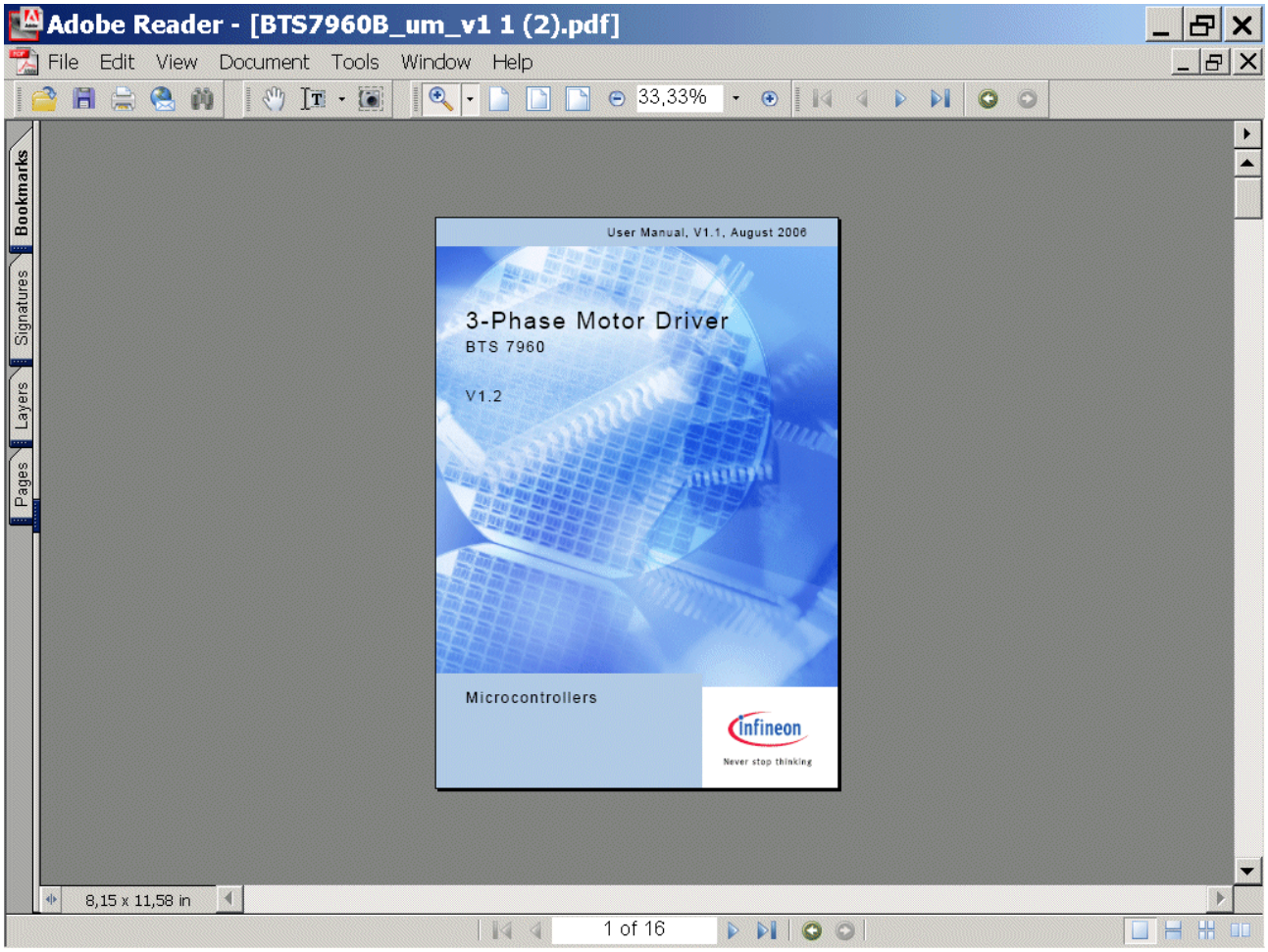





Shunt
To ADC_Cannel_3

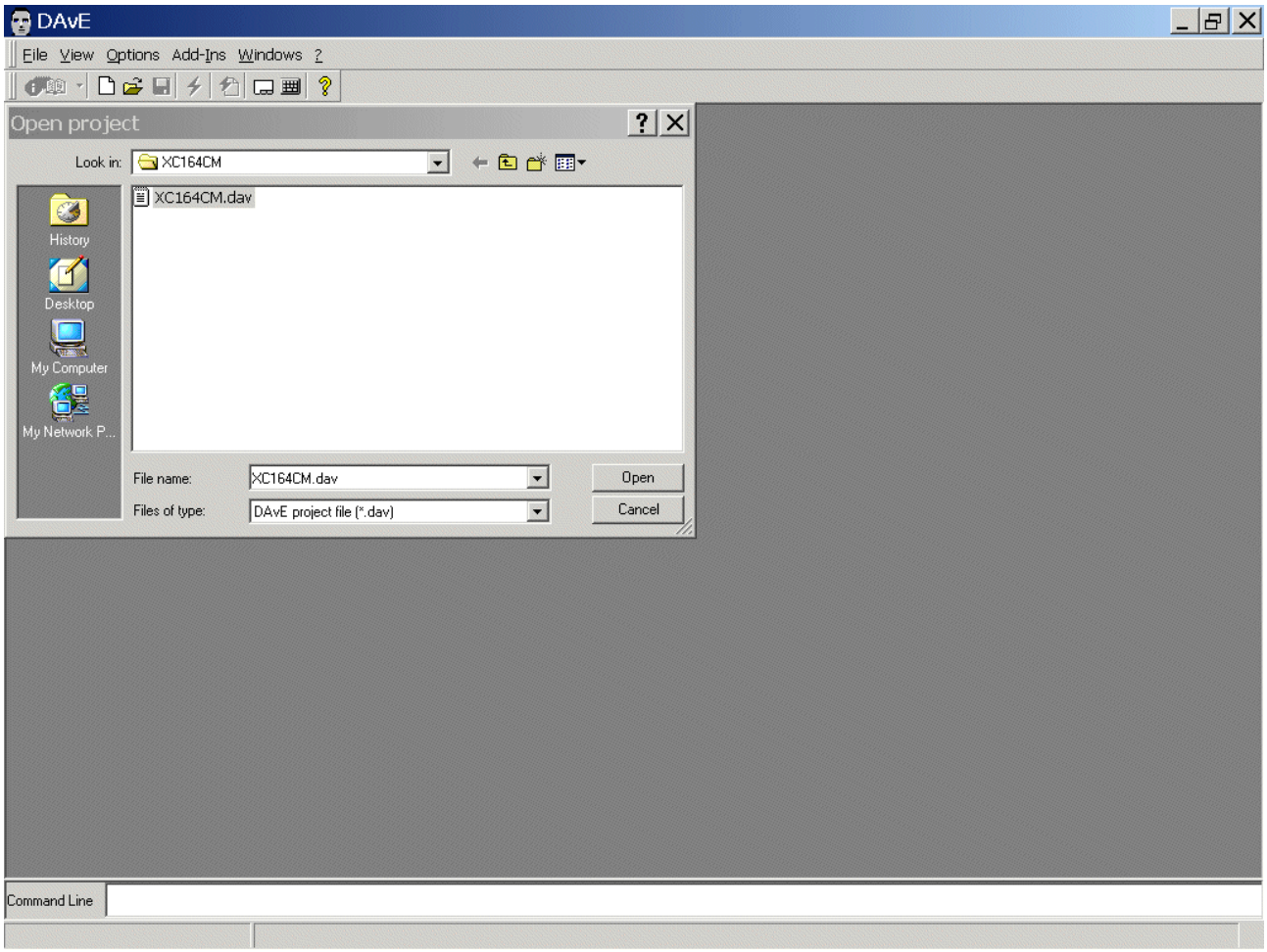

Adobe Reader - [BTS7960B_um_v1.1.pdf]
File Edit View Document Tools Window Help
44%
Options
Bookmarks
Signatures
Layers
Pages
1 Overview
1.1 Features
1.2 Electrical Characteristics
2 Board Description
2.1 Connectors and Signals
2.1.1 Power Supply Connector X3
2.1.2 Motor Output Connector X4
2.1.3 Hall Interface Connector J1
2.1.4 Easy Kit Connectors
2.2 Mounting Options
2.2.1 Current Sense / Shunt
2.2.2 Hall Sensor Supply
2.2.3 CTRAP Disable
2.2.4 Slew-Rate Adjustment
3 Schematics
4 PCB - Layout
5 Bill of Material
3-Phase Motor Driver
BTS 7960
Board Description
2.2 Mounting Options
2.2.1 Current Sense / Shunt
The 3-Phase Motor Driver provides two ways of current measurement via ADC channel 3. There is a shunt resistor and an amplifier available, or the current sense signal of BTS 7960B can be used.
Shunt (default)
By mounting R24 (1 kΩ) and the shunt R34 (0.022 Ω), the current through the motor can be measured by ADC channel 3 with a ratio of 1.78 A/V.
Current Sense
By unmounting R24 and mounting R29 (0 Ω) and R30 (4.7 kΩ), the same ratio can be acceived.
2.2.2 Hall Sensor Supply
There is a mounting option to supply the hall sensors either by VBB (default) or by VCC. To change to VCC supply, unmount R20 and mount R19 (0 Ω).
2.2.3 CTRAP Disable
The CTRAP feature is enabled per default. To disable the CTRAP on hardware level, unmount R25 (0 Ω).
Attention: The CTRAP protects the hardware, when the software intends to drive too much current. When disabled, the hardware might be destructed.
2.2.4 Slew-Rate Adjustment
The BTS 7960B provides an adjustable slew-rate. Please refer to the data sheet of this power device for details. The dedicated resistors for slew-rate adjustment are R31, R32 and R33.
User Manual 10 V1.1, 2006-08-10
8,15 x 11,58 in
10 of 16

Note:
For further information, please refer to the 3-Phase Motor Driver BTS 7960 User Manual.

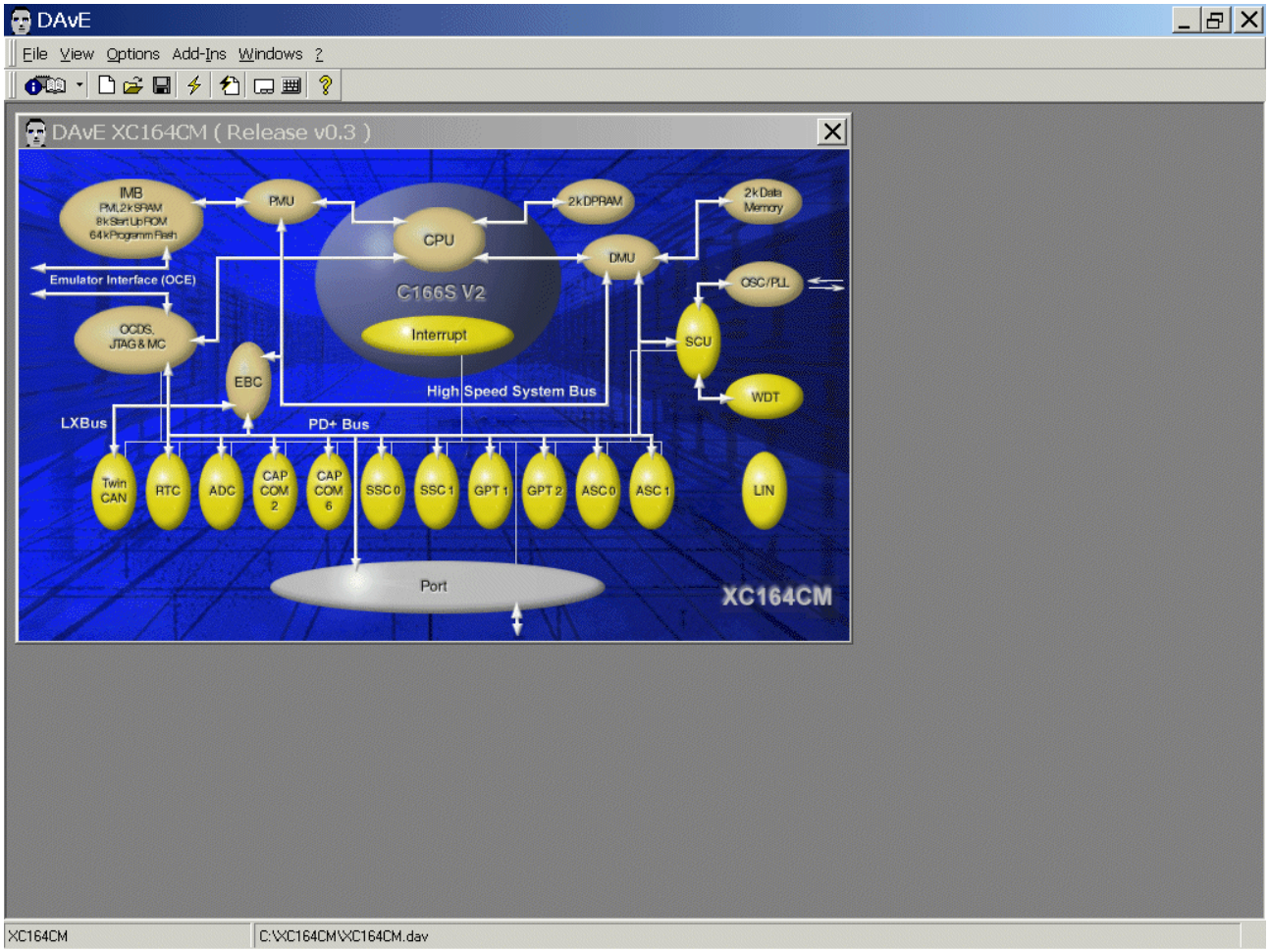


Start/Goto the program generator DAvE and open your XC164CM.dav DAvE project:

File
Open
Location: C:\XC164CM
Filename: XC164CM.dav

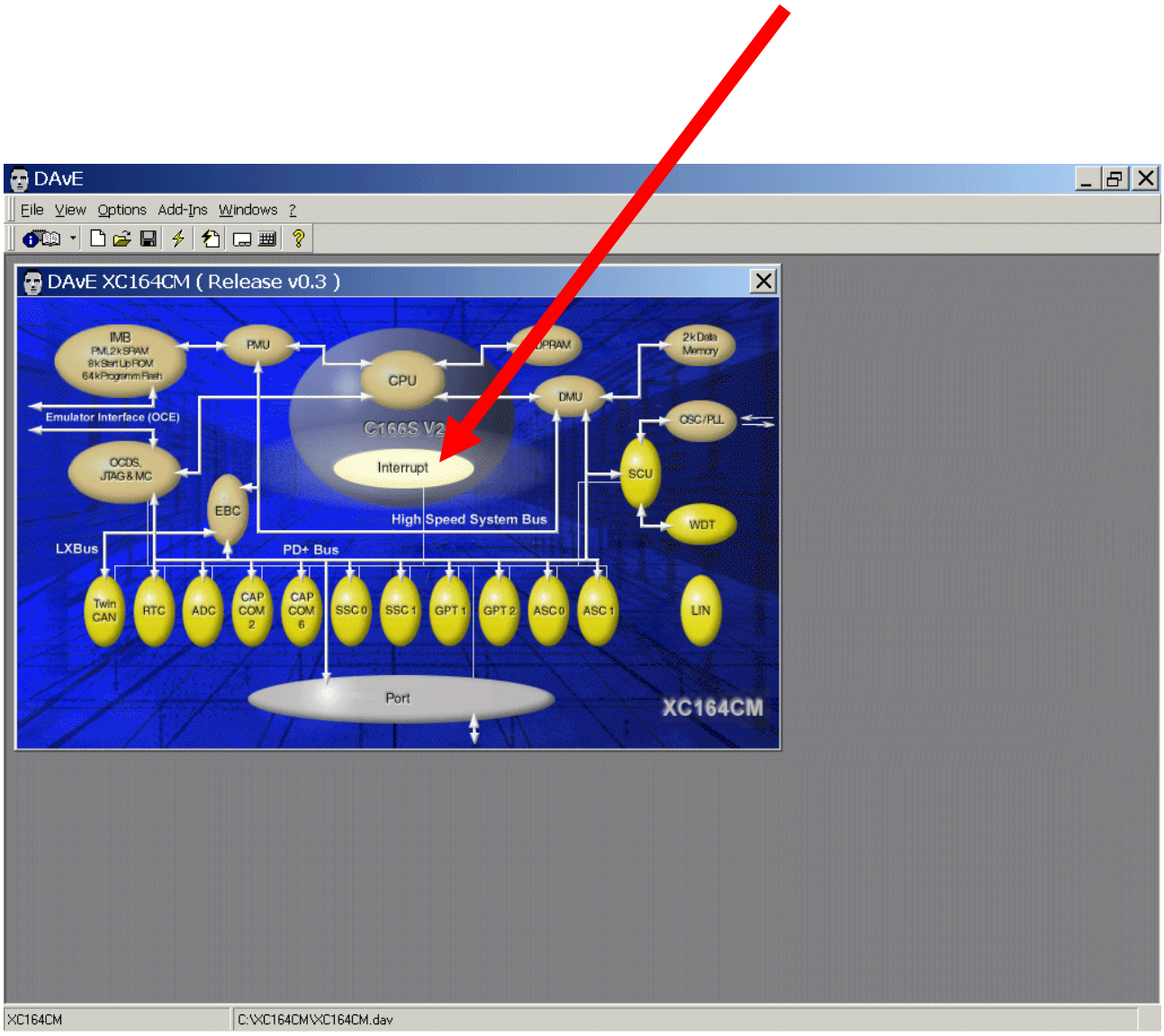


Click Open


DAvE
File View Options Add-Ins Windows ?
DAvE XC164CM (Release v0.3)
IMB
PMU
CPU
2k DPRAM
2k Data Memory
DMU
C166S V2
Interrupt
Emulator Interface (OCE)
OCDS, JTAG & MC
EBC
OSC/PLL
SCU
WDT
High Speed System Bus
LXBus
PD+ Bus
Twin CAN
RTC
ADC
CAP COM 2
CAP COM 6
SSC 0
SSC 1
GPT 1
GPT 2
ASC 0
ASC 1
LIN
Port
XC164CM
XC164CM
C:\XC164CM\XC164CM.dav

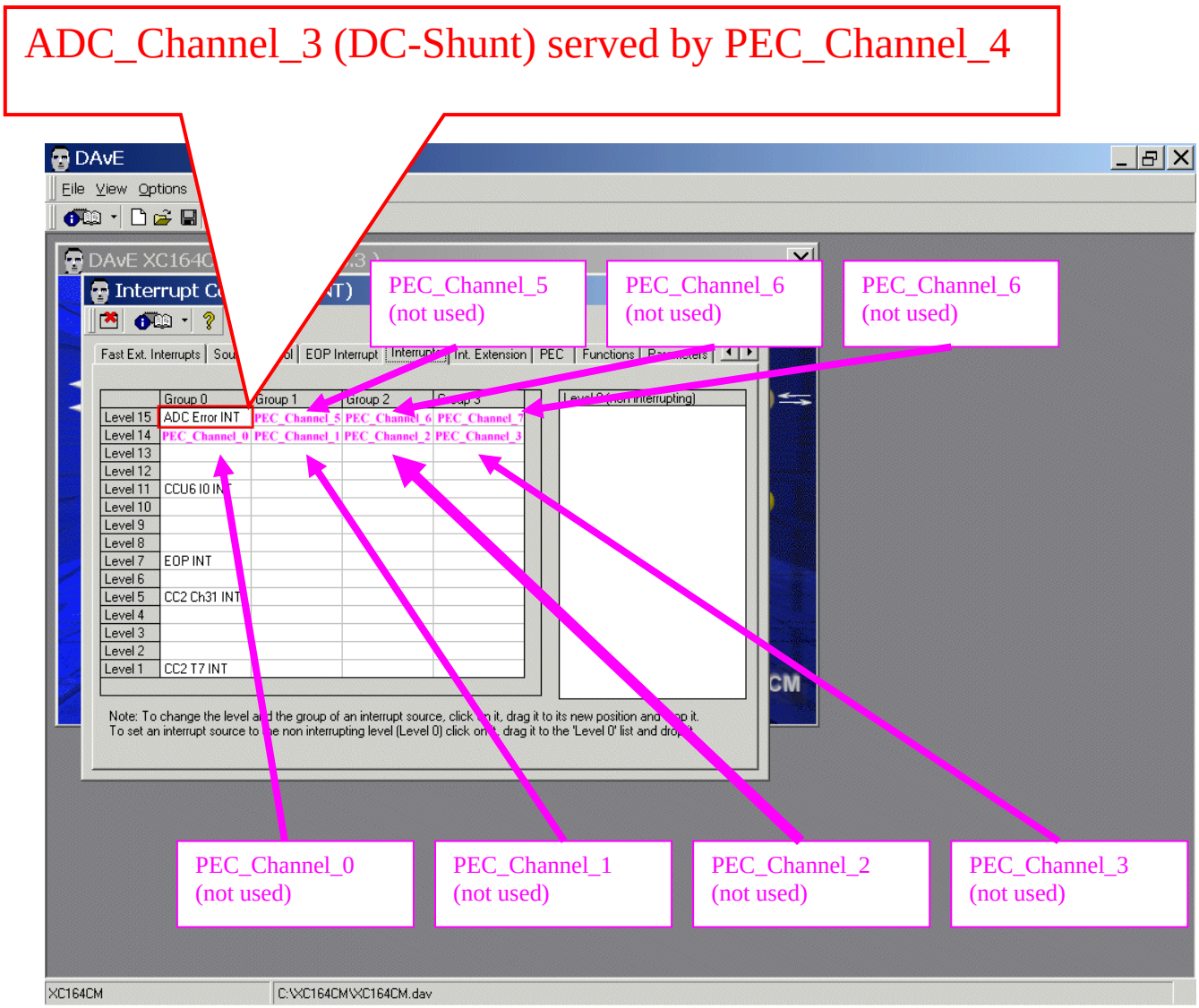
Configuration of the Interrupt-System (configure as you can see in the screenshots):

The configuration window can be opened by clicking the specific block/module.

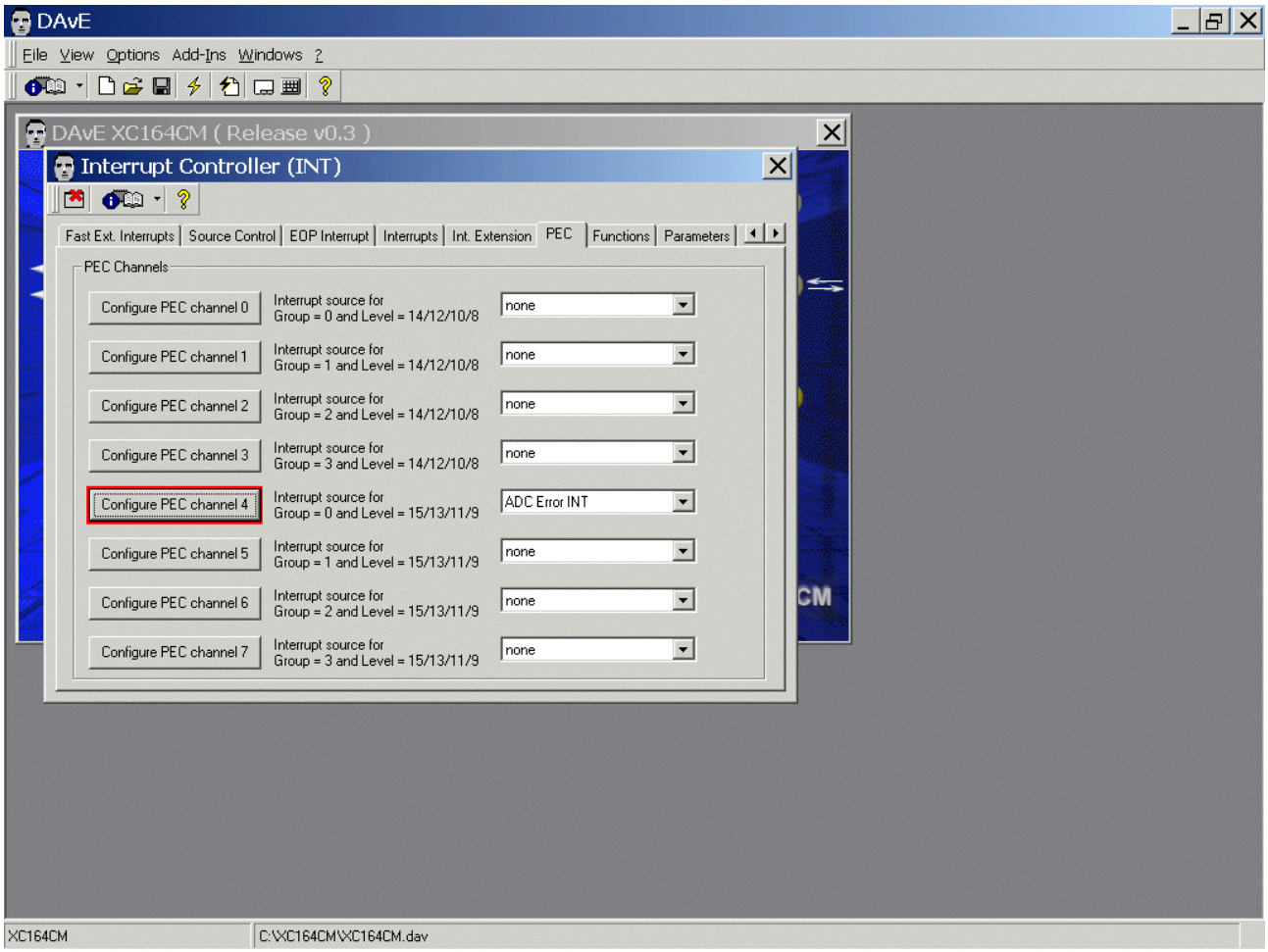


Overview PEC Channels:

INT: Interrupts :



INT: PEC: PEC Channels: click Configure PEC channel 4



INT: PEC: PEC Channels: click Configure PEC channel 4

Transfer Count: click ⊙ Decrement counter

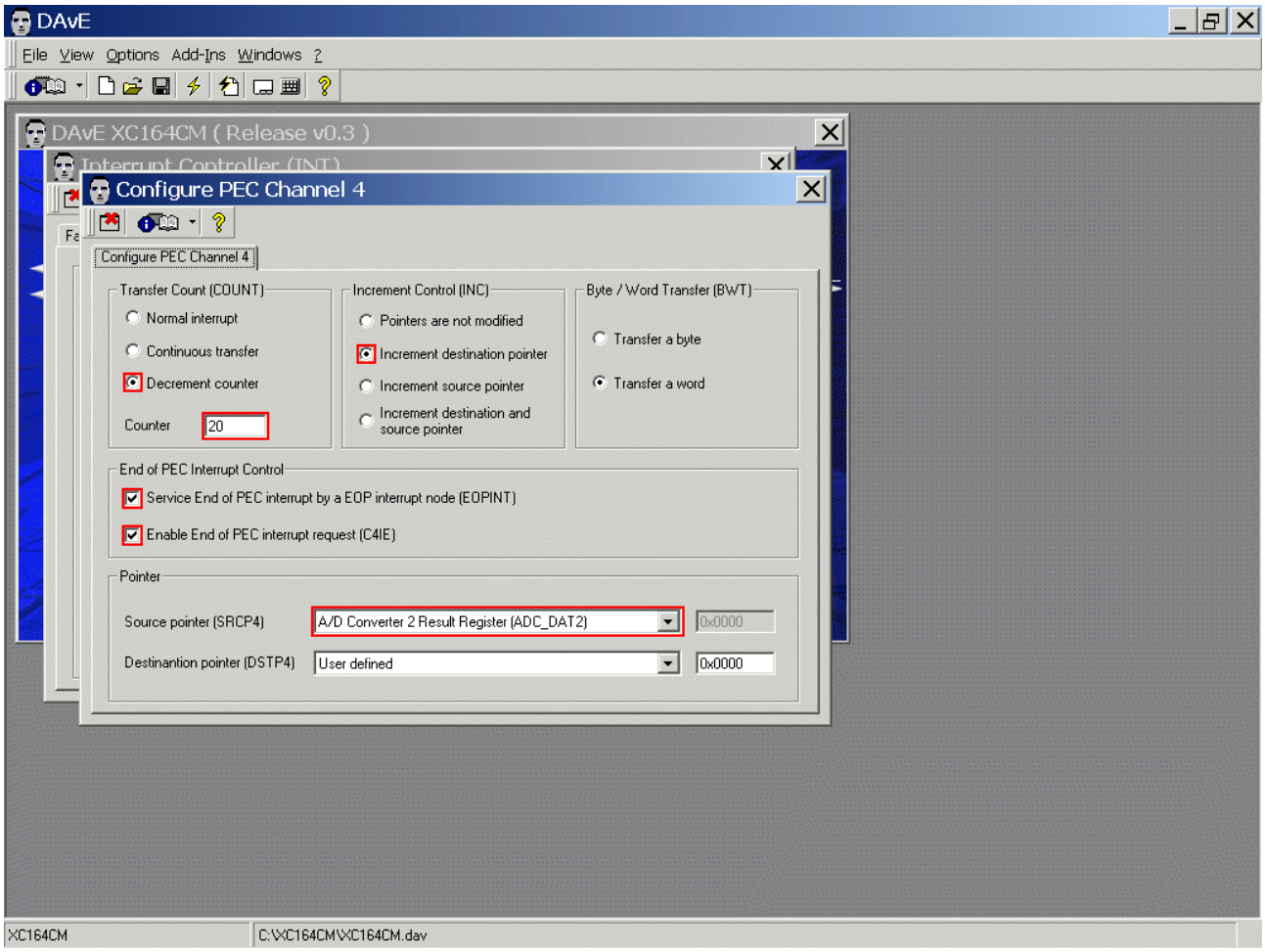
Transfer Count: insert 20 <ENTER>

Increment Control: click ⊙ Increment destination pointer

End of PEC Interrupt Control: click ✓ Service End of PEC interrupt by a EOP interrupt node

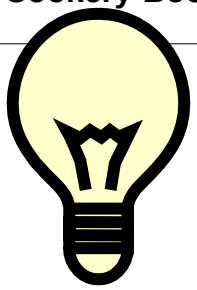
End of PEC Interrupt Control: click ✓ Enable End of PEC interrupt request

Pointer: Source pointer: select A/D Converter 2 Result Register (ADC_DAT2)



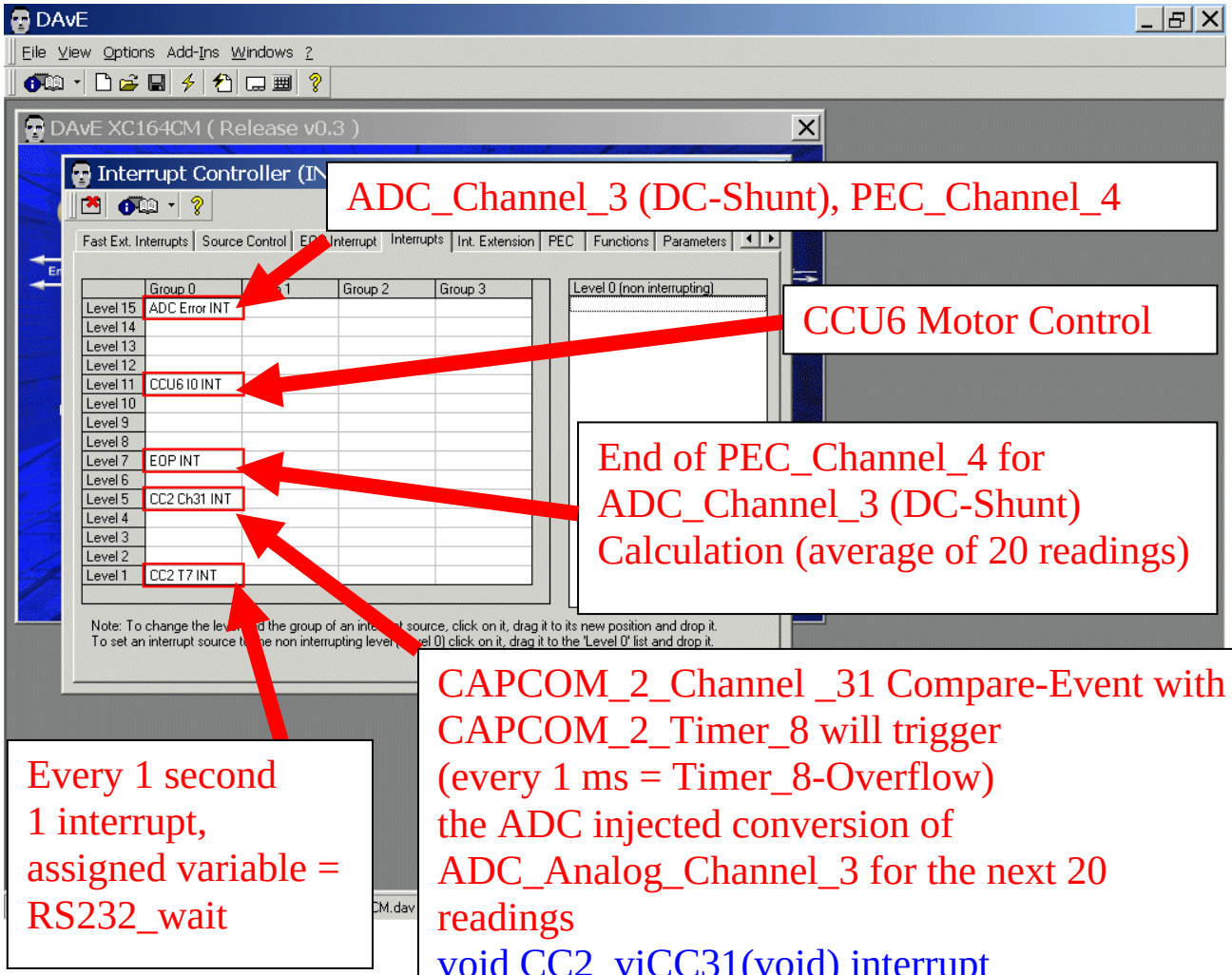
Exit this dialog now by clicking the close button.

Exit this dialog now by clicking the close button.



Remember :

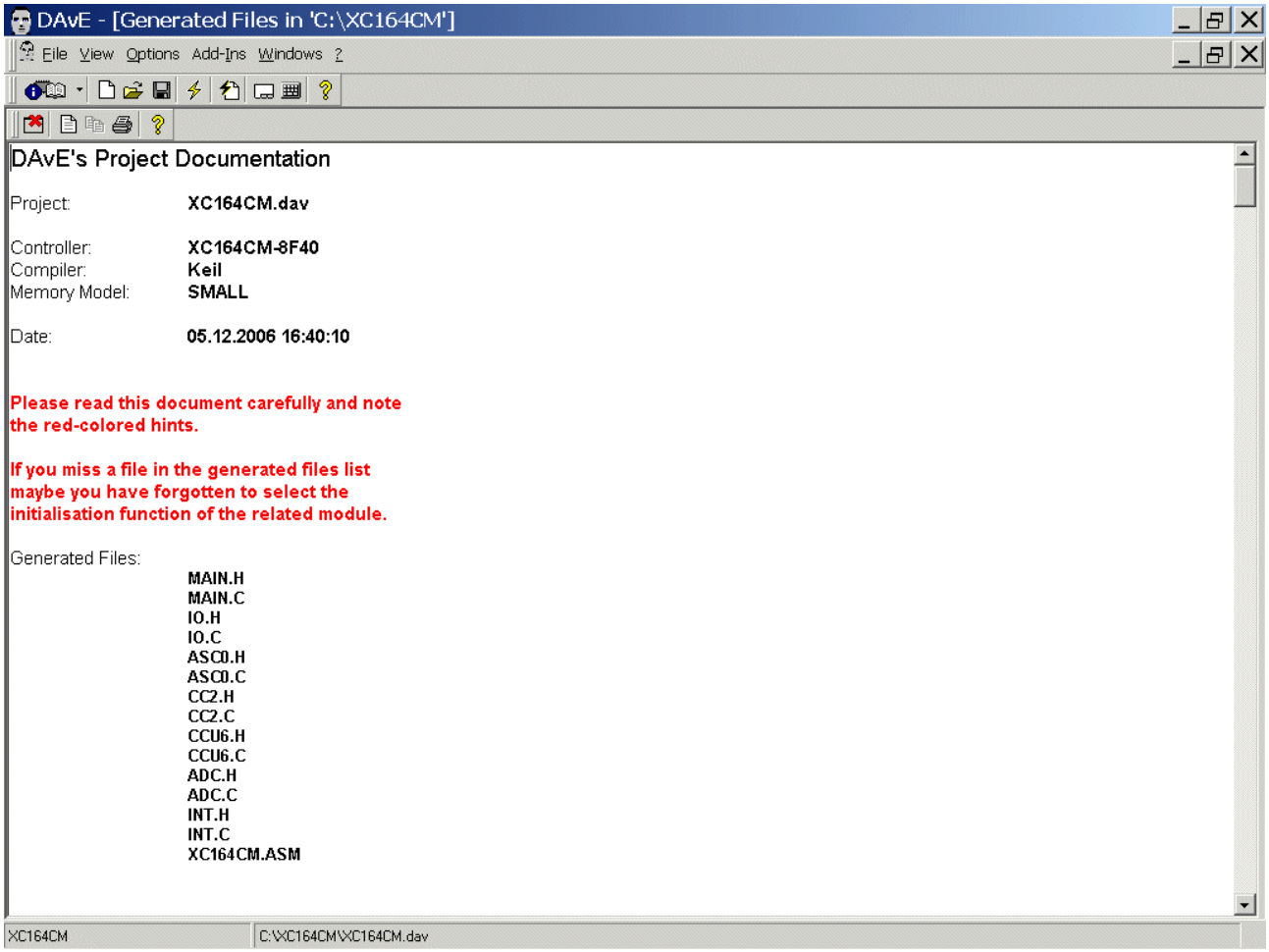
Current Configuration of the Interrupt System:



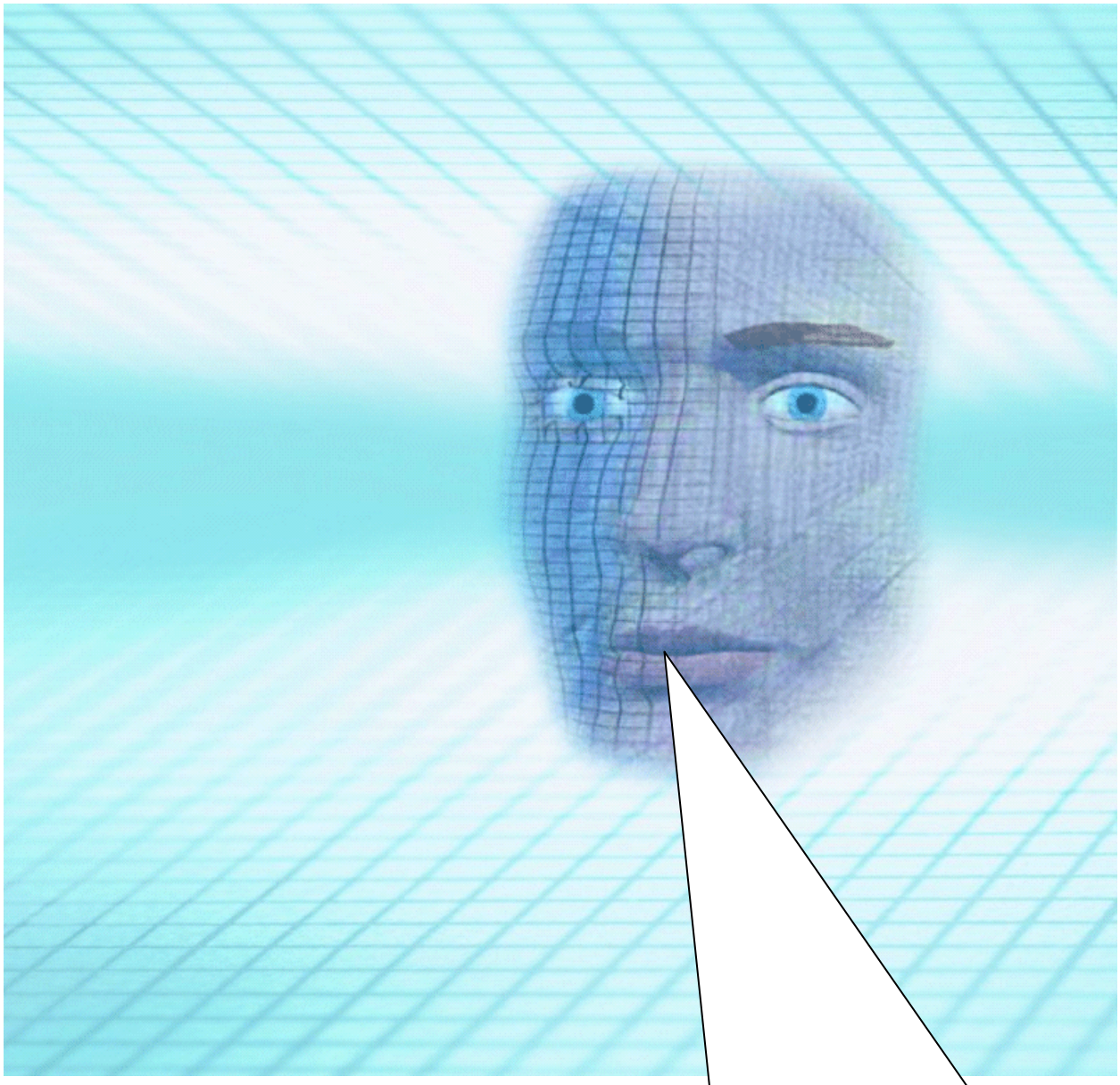
Generate Code:

File Generate Code	or click 

DAvE will show you all the files he has generated
(File Viewer opens automatically).

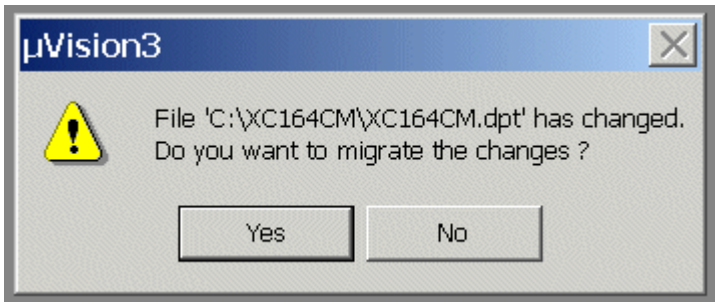


Insert your application specific program:



Note:
DAvE doesn't change code which is inserted between '// USER CODE BEGIN' and '// USER CODE END'. Therefore, whenever adding code to DAvE's generated code, write it between '// USER CODE BEGIN' and '// USER CODE END'.
If you wish to change DAvE´s generated code or add code outside these 'USER CODE' sections you will have to insert/modify your changes each time after letting DAvE regenerate code!

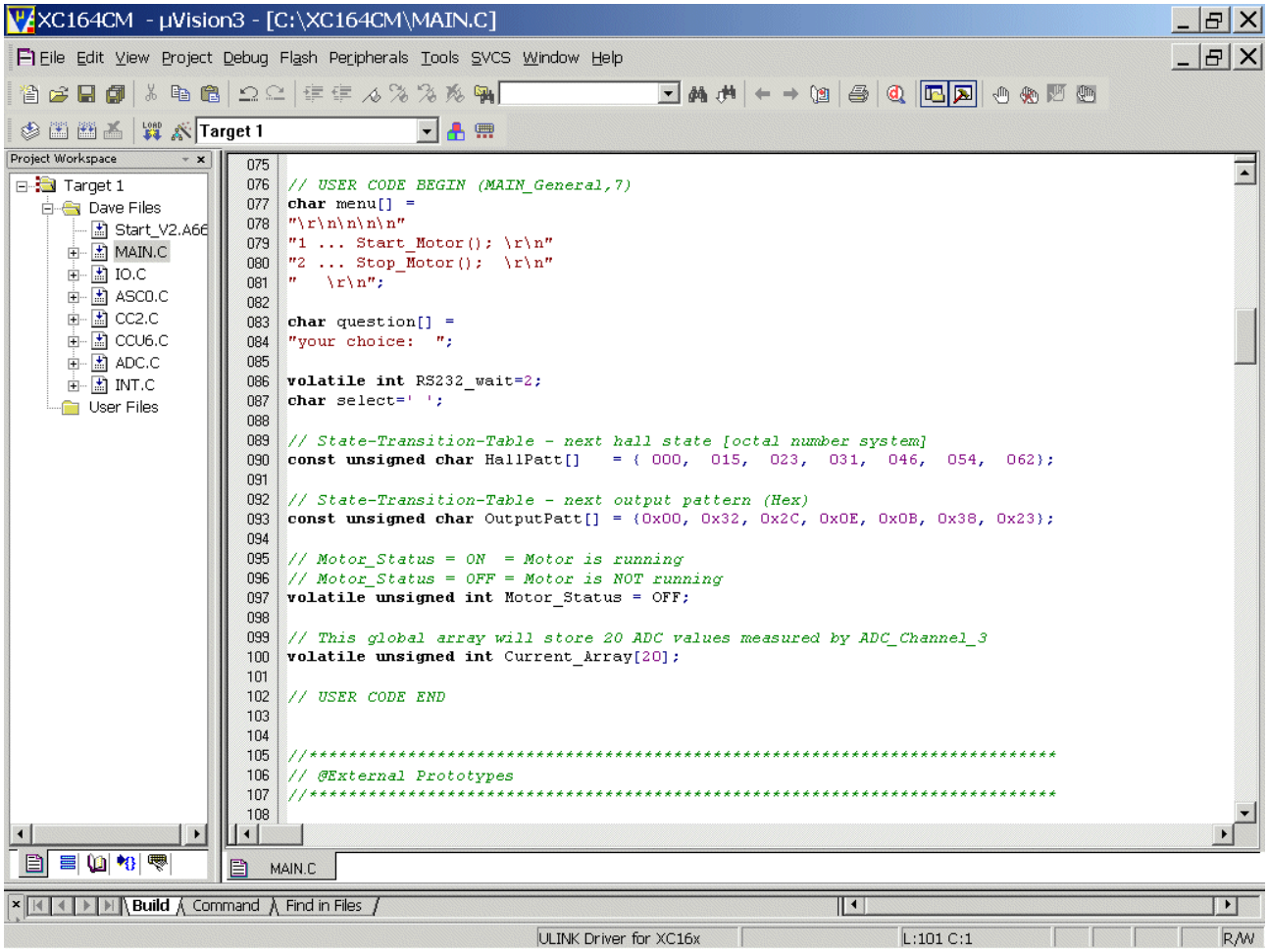
Goto µVision:



Click Yes

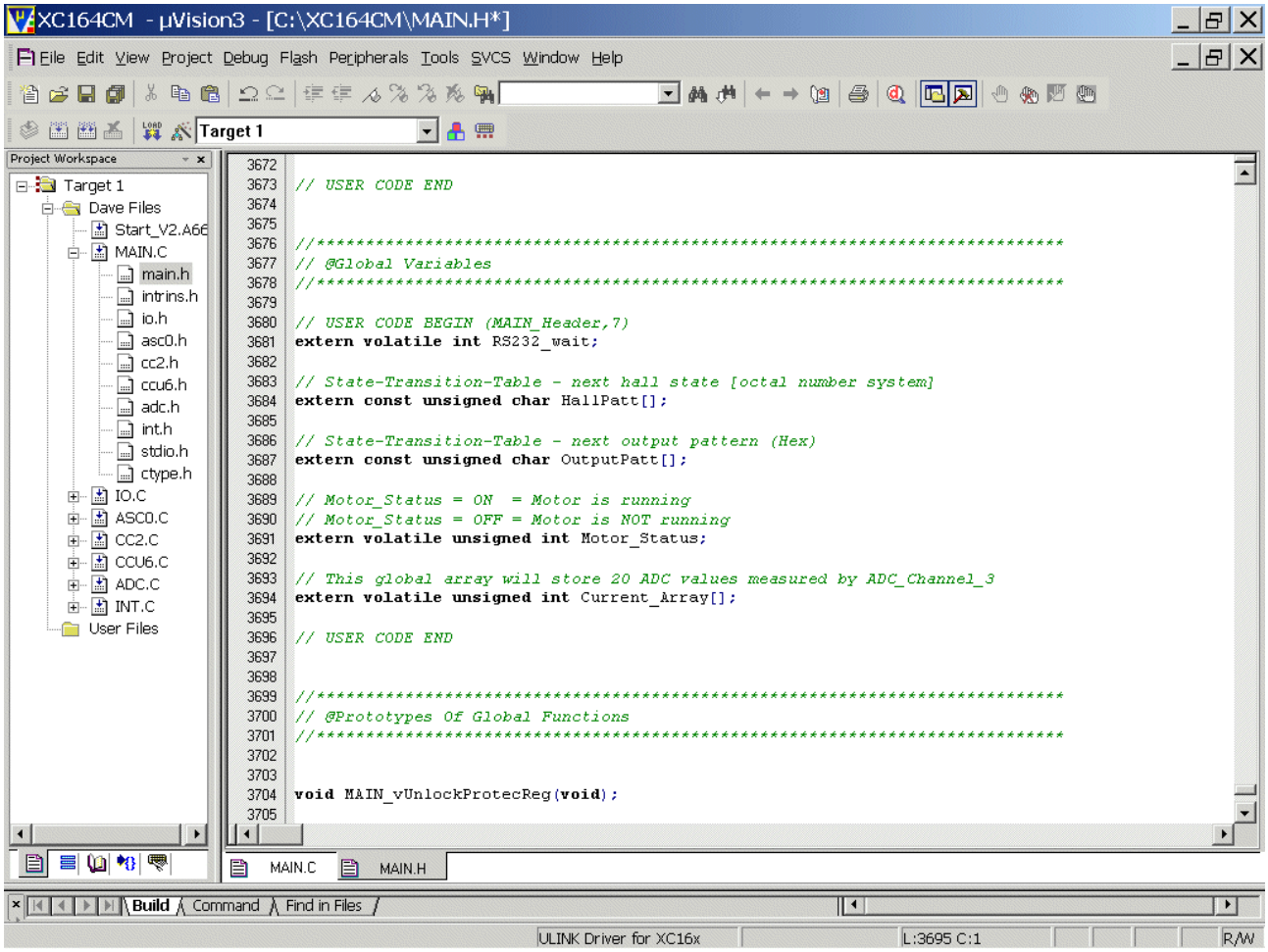
Double click MAIN.C and insert Global Variables:

```
// This global array will store 20 ADC values measured by
ADC_Channel_3
volatile unsigned int Current_Array[20];
```



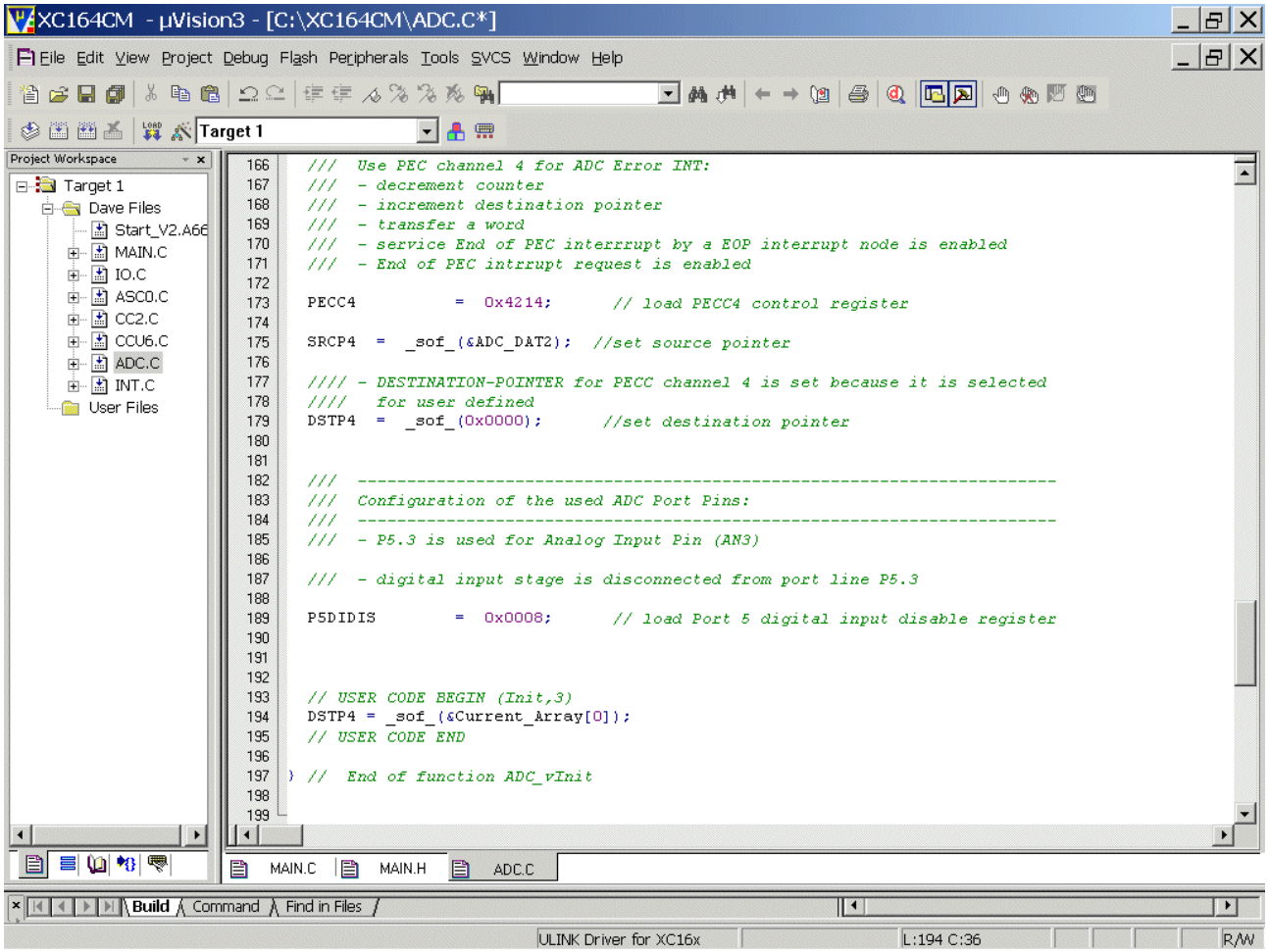
Double click main.h and insert Extern Declaration of Global Variable:

```
// This global array will store 20 ADC values measured by
ADC_Channel_3
extern volatile unsigned int Current Array[];
```



Double click ADC.C and insert Code:

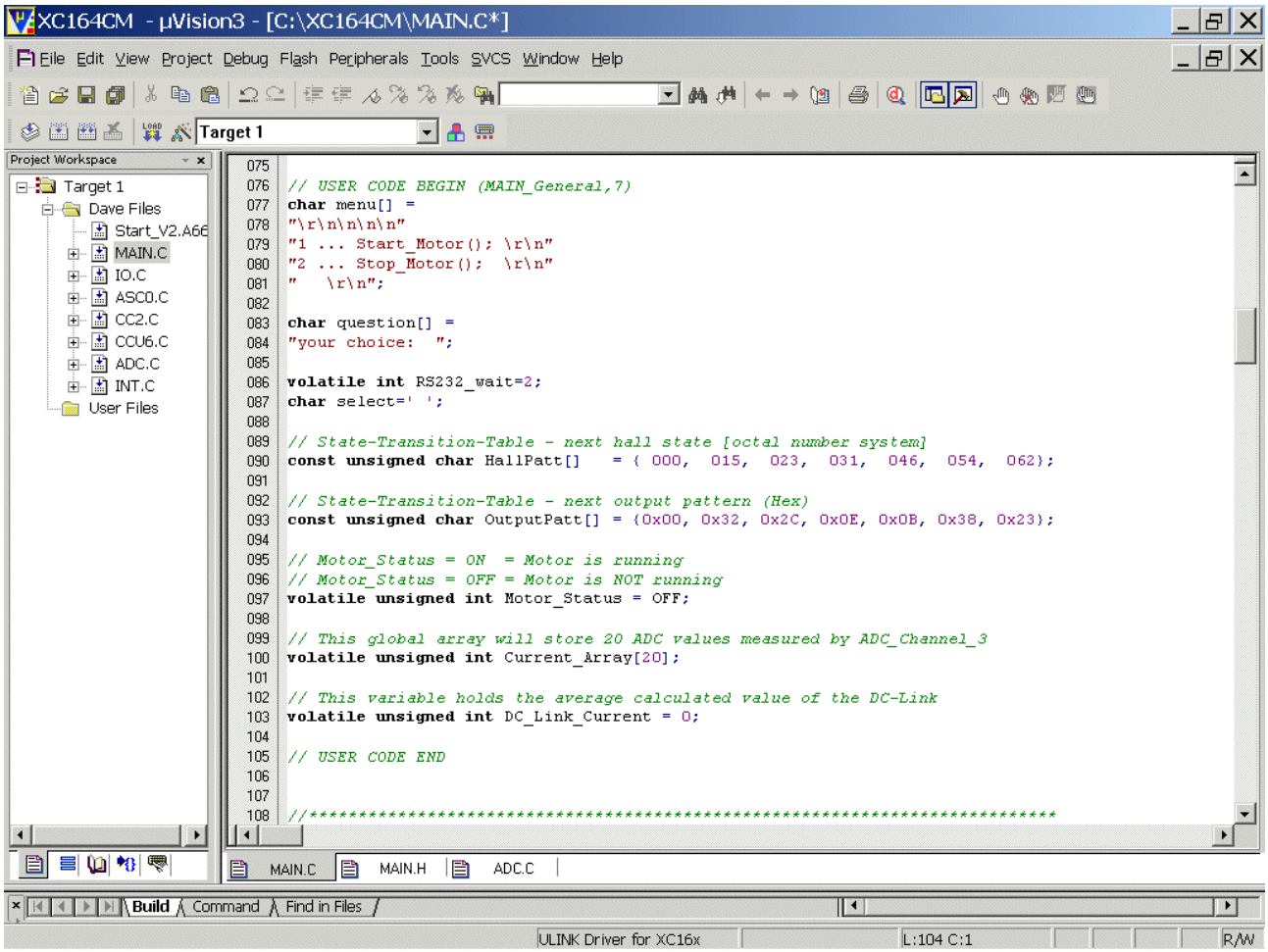
```
DSTP4 =  sof (&Current Array[0]);
```



Create a new global variable which holds the average calculated value of the DC-Link:

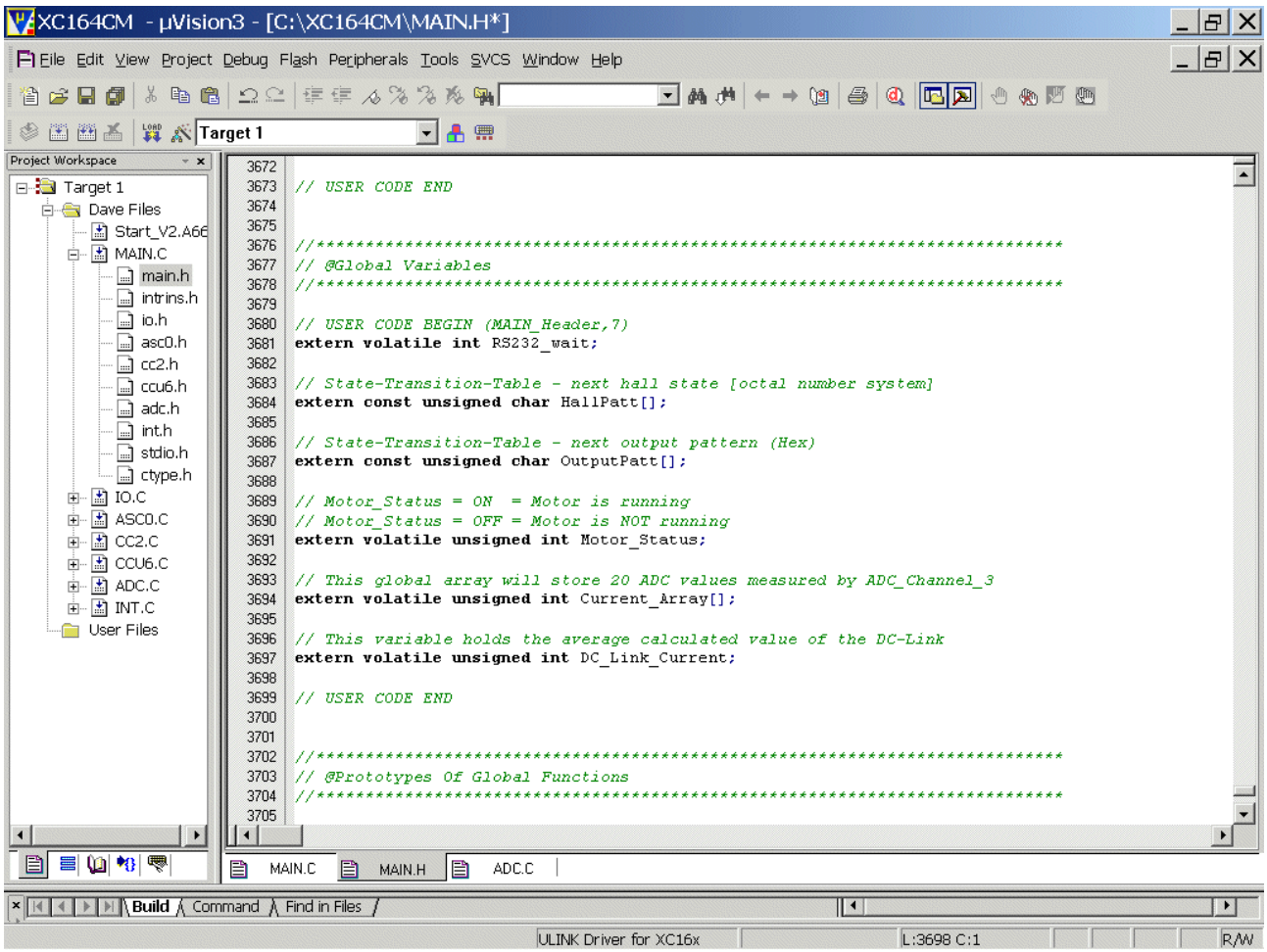
Double click MAIN.C and insert Global Variables:

```
// This variable holds the average calculated value of the DC-Link
volatile unsigned int DC_Link_Current = 0;
```



Double click main.h and insert Extern Declaration of Global Variable:

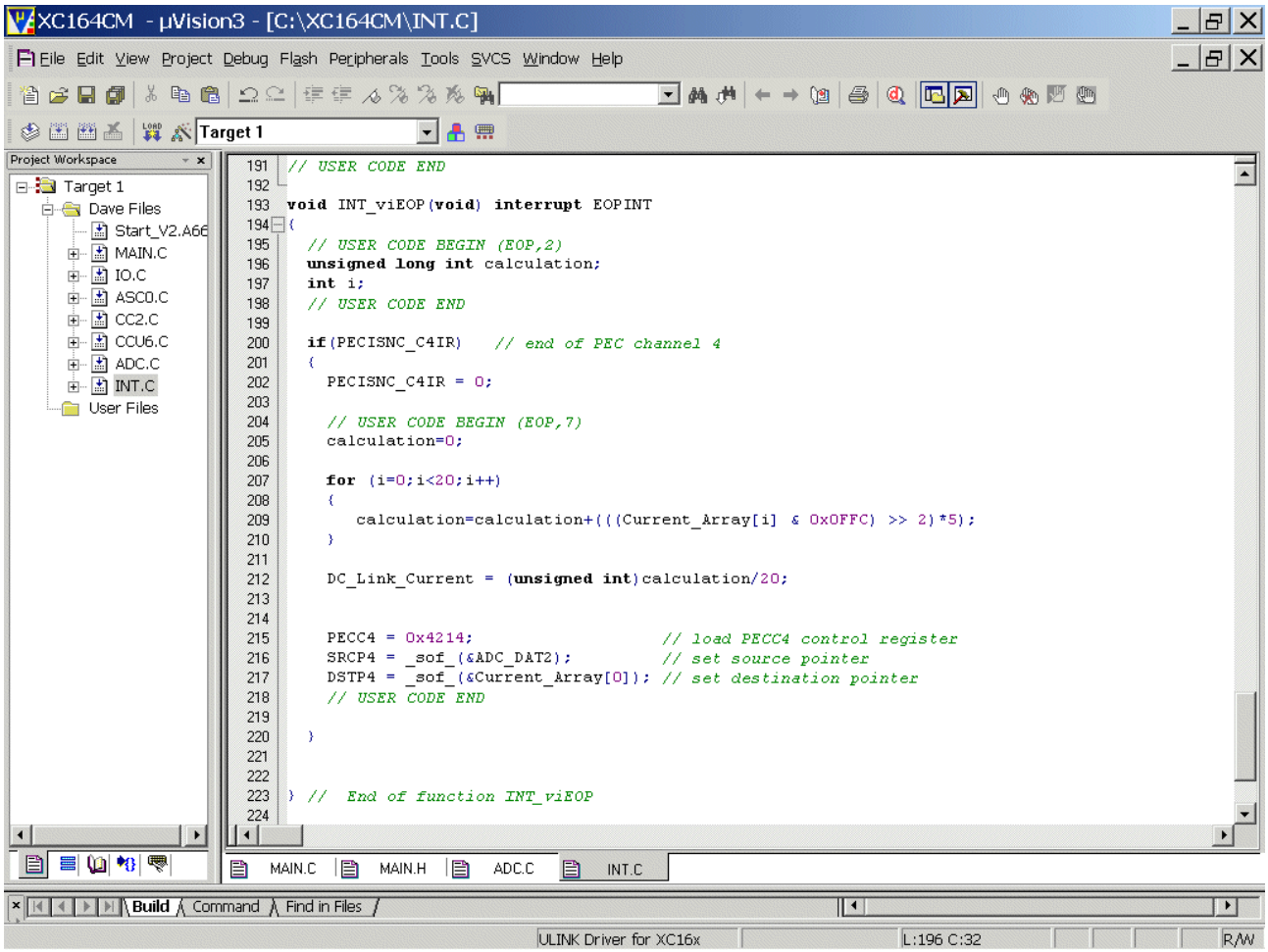
```
// This variable holds the average calculated value of the DC-Link
extern volatile unsigned int DC_Link_Current;
```

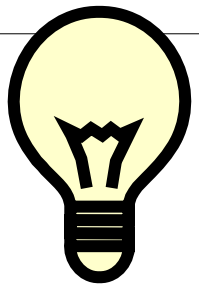


Double click INT.C and insert Code:

```
unsigned long int calculation;
int i;
```

```
    calculation=0;

    for (i=0;i<20;i++)
    {
       calculation=calculation+(((Current_Array[i] & 0x0FFC) >> 2)*5);
    }

    DC_Link_Current = (unsigned int)calculation/20;


    PECC4 = 0x4214;                    // load PECC4 control register
    SRCP4 = _sof_(&ADC_DAT2);          // set source pointer
    DSTP4 = _sof_(&Current_Array[0]); // set destination pointer
```





Register Information / Additional Information:

Adobe Reader - [xc164_um_v1.2_2006_03_per.pdf]

File Edit View Document Tools Window Help

ADC_DAT
ADC Result Register **SFR ($FEA0_H$/50_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				ADRES											
rwh				rwh											

ADC_DAT2
ADC Chan. Inj. Result Reg. **ESFR ($F0A0_H$/50_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				ADRES											
rw				rwh											

Field	Bits	Type	Description
CHNR	[15:12]	rw[h]	**Channel Number** (identifies the converted analog channel)
ADRES	[11:0]	rwh	**A/D Conversion Result** The digital result of the most recent conversion. In compatibility mode, the result is placed as follows: 8-bit: ADRES[9:2] 10-bit: ADRES[9:0] In enhanced mode, the result is placed as follows: 8-bit: ADRES[11:4] 10-bit: ADRES[11:2] *Note: Unused bits of ADRES are always set to 0.*

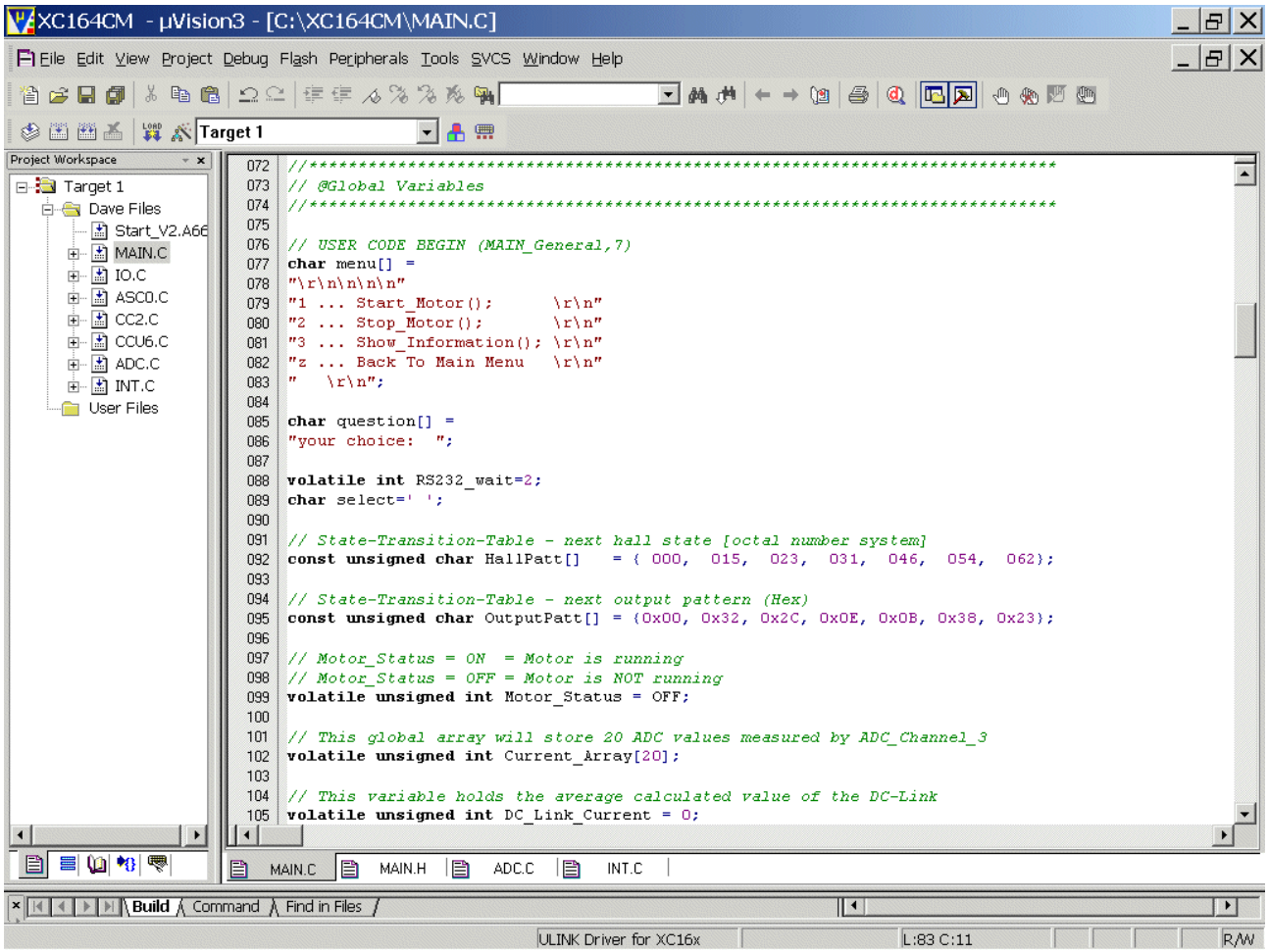
8,15 x 11,58 in 92 of 412

Double click MAIN.C and change Global Variable from:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor(); \r\n"
"2 ... Stop_Motor();  \r\n"
"   \r\n";
```

to:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor();       \r\n"
"2 ... Stop_Motor();        \r\n"
"3 ... Show_Information(); \r\n"
"z ... Back To Main Menu    \r\n"
"   \r\n";
```

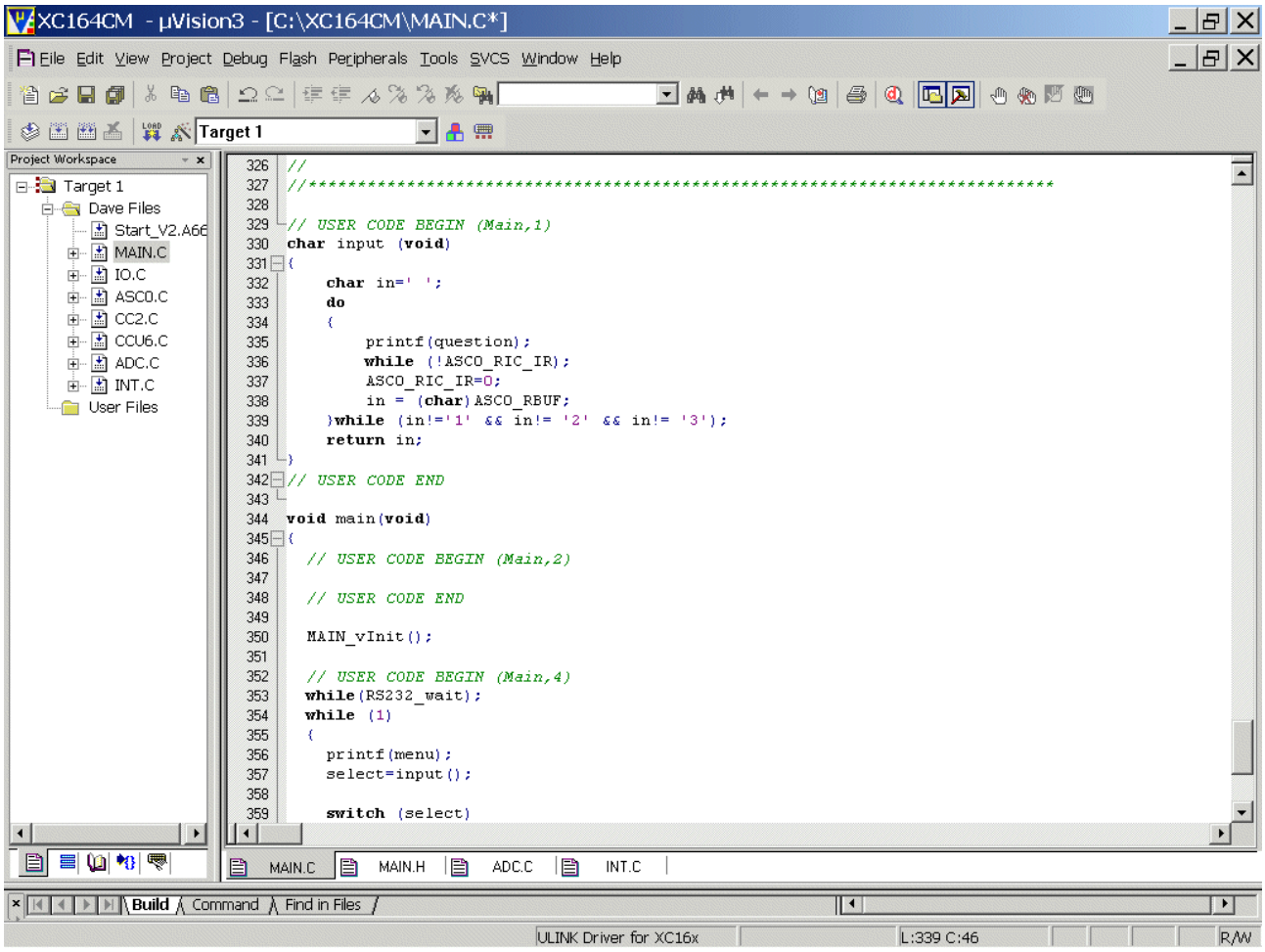


Double click MAIN.C and change Code from:

```
}while (in!='1' && in!= '2');
```

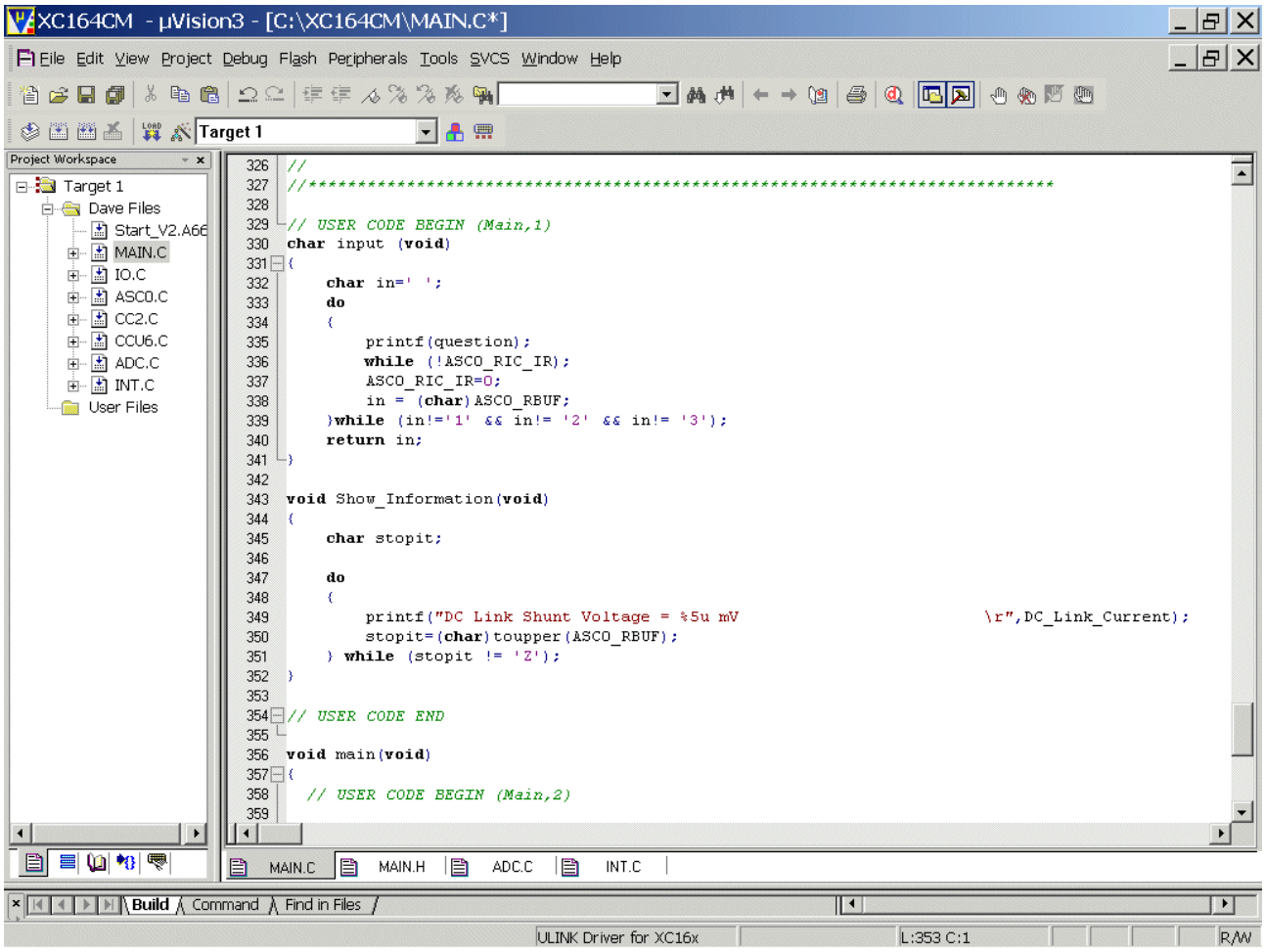
to:

```
}while (in!='1' && in!= '2' && in!= '3');
```



Double click MAIN.C and insert Code:

```
void Show_Information(void)
{
    char stopit;

    do
    {
        printf("DC Link Shunt Voltage = %5u mV
\r",DC_Link_Current);
        stopit=(char)toupper(ASC0_RBUF);
    } while (stopit != 'Z');
}
```

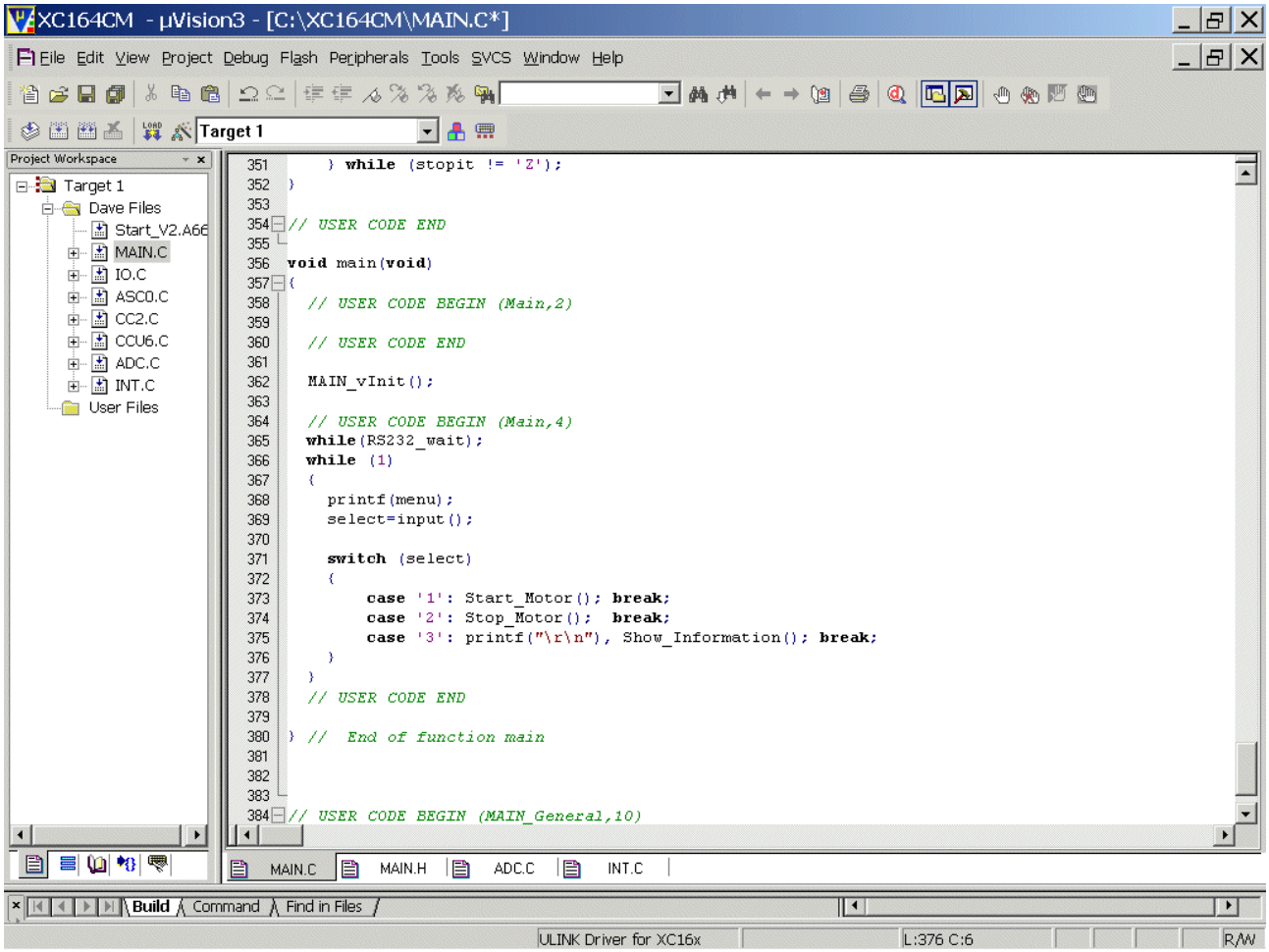


Double click MAIN.C and change Code from:

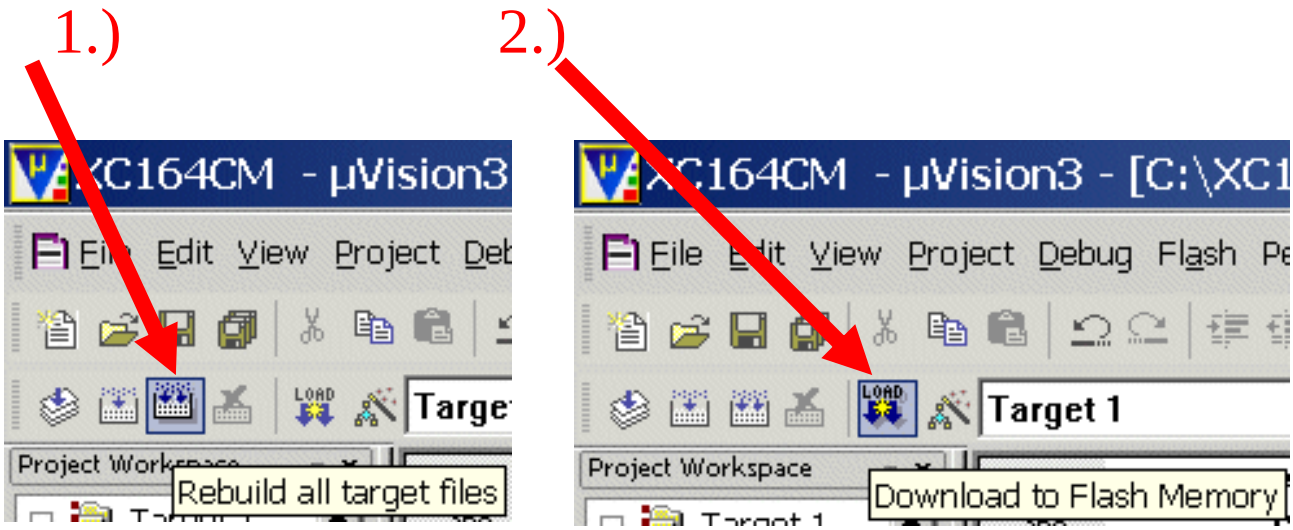
```
switch (select)
{
      case '1': Start_Motor(); break;
      case '2': Stop_Motor(); break;
}
```

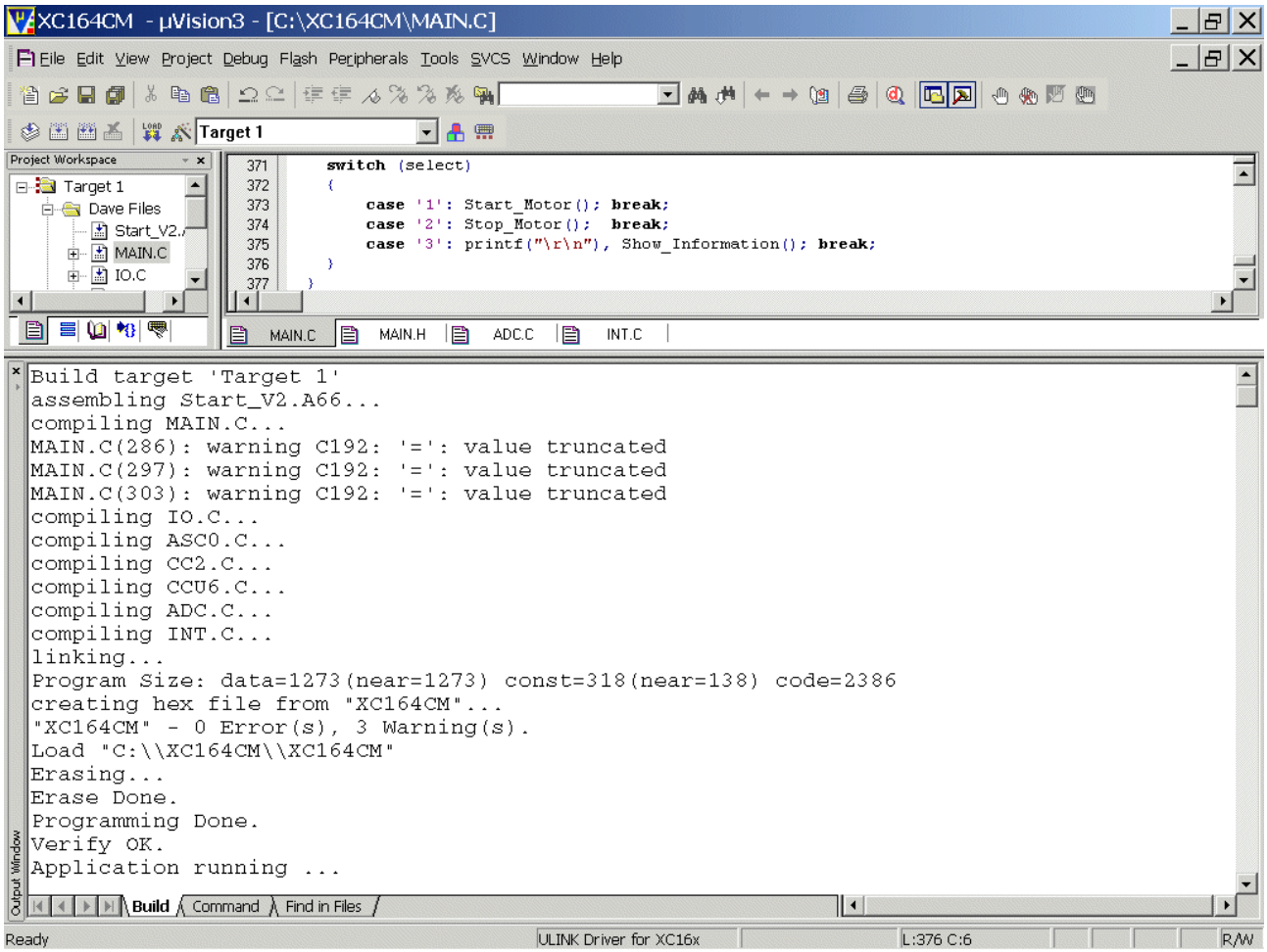
to:

```
switch (select)
{
    case '1': Start_Motor(); break;
    case '2': Stop_Motor();  break;
    case '3': printf("\r\n"), Show_Information(); break;
}
```

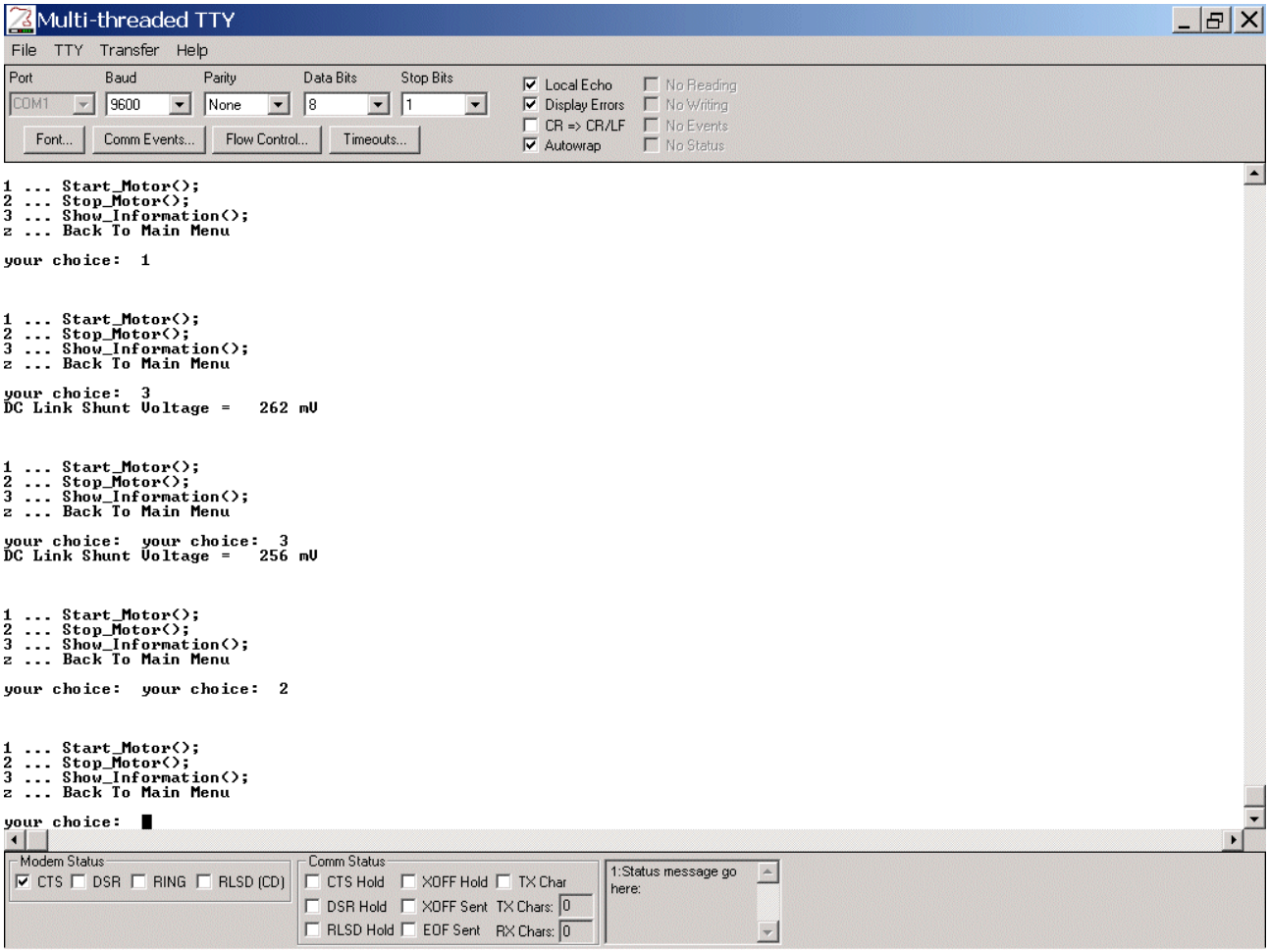


Build Application:





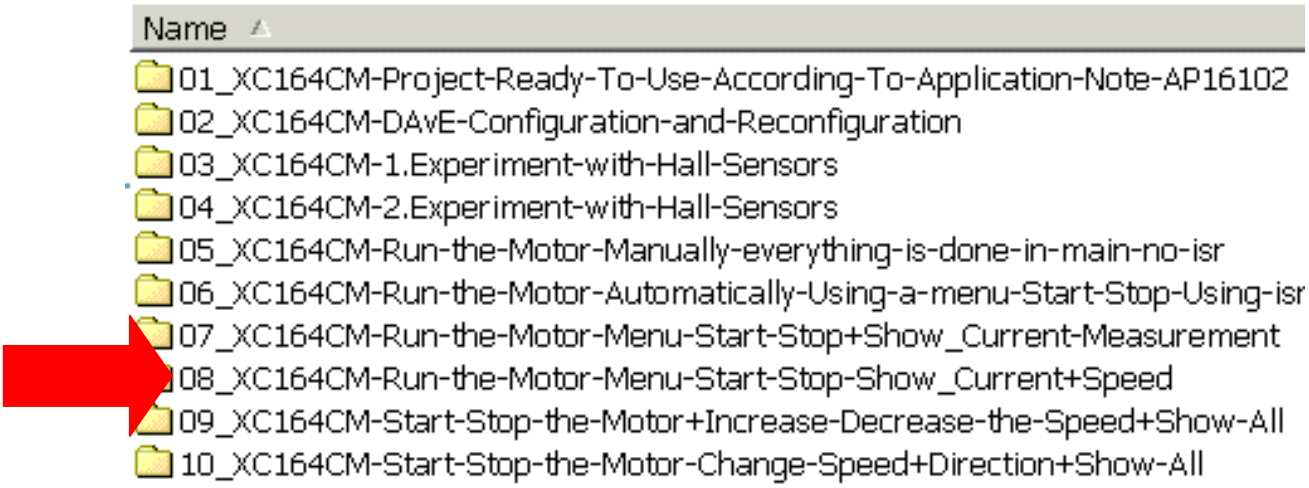
And see the result:

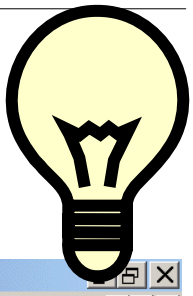


8.)

Calculating The Speed Of The Motor:

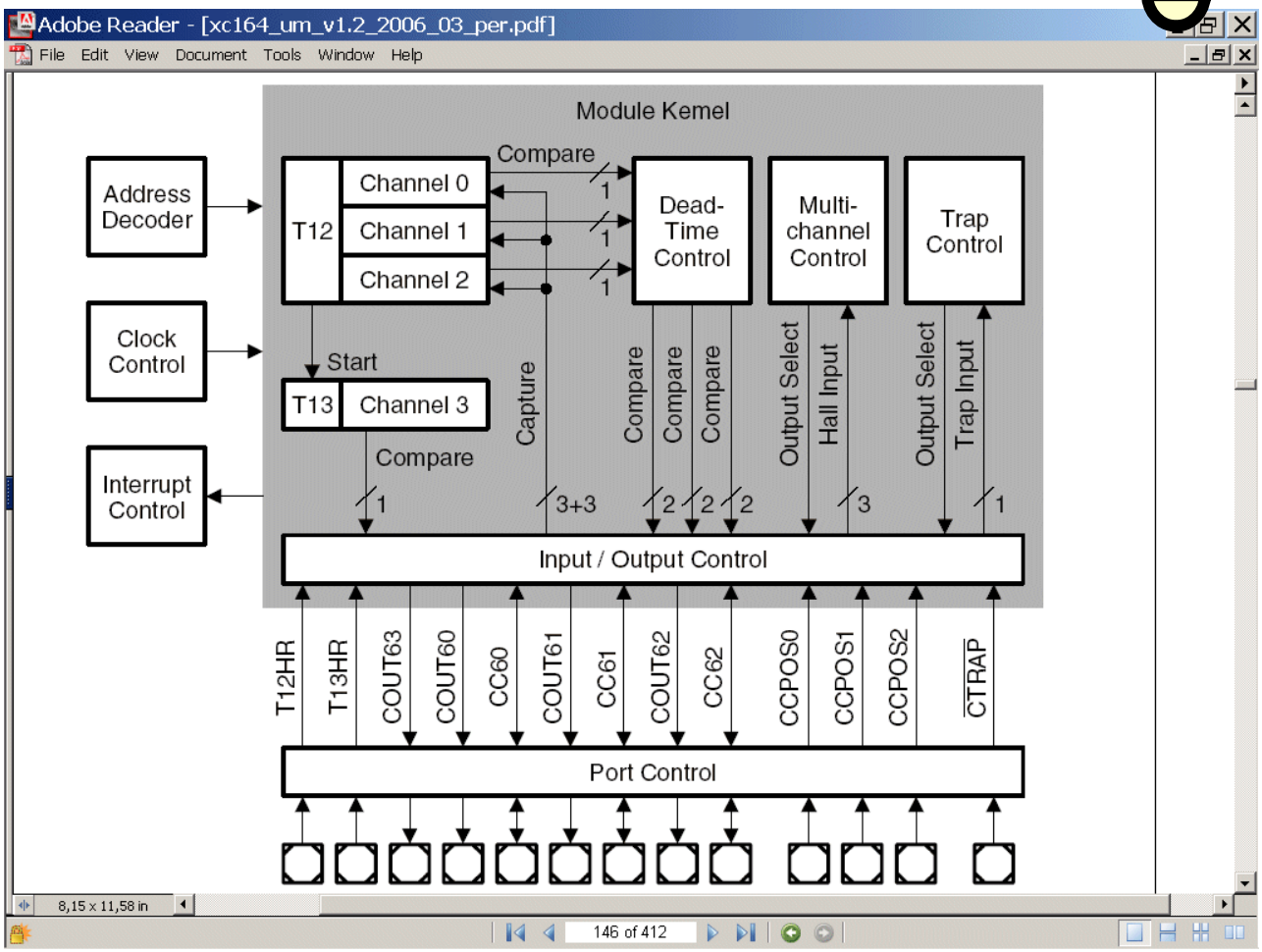
Relevant Project:



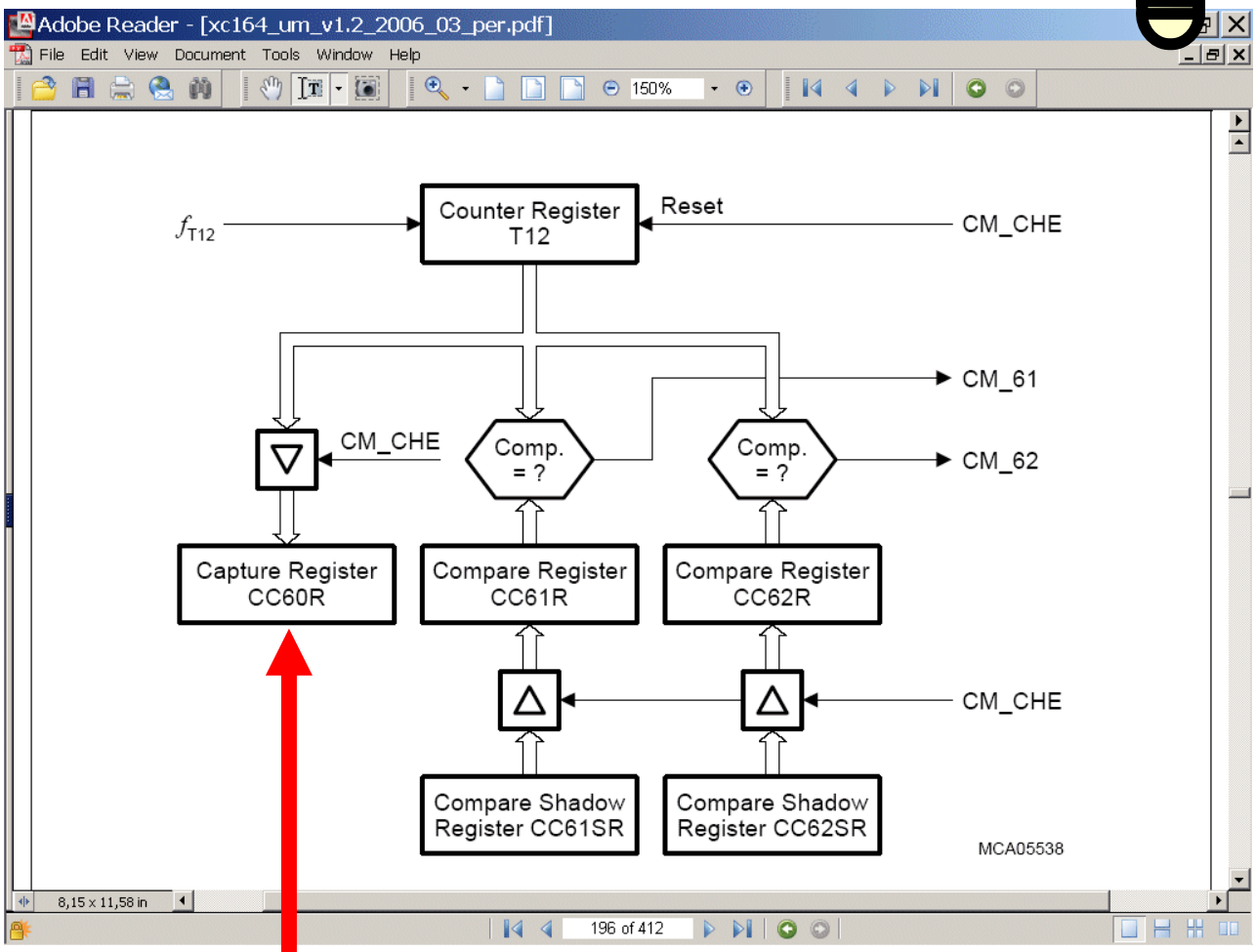


Register Information / Additional Information:

CAPCOM6 Block Diagram:

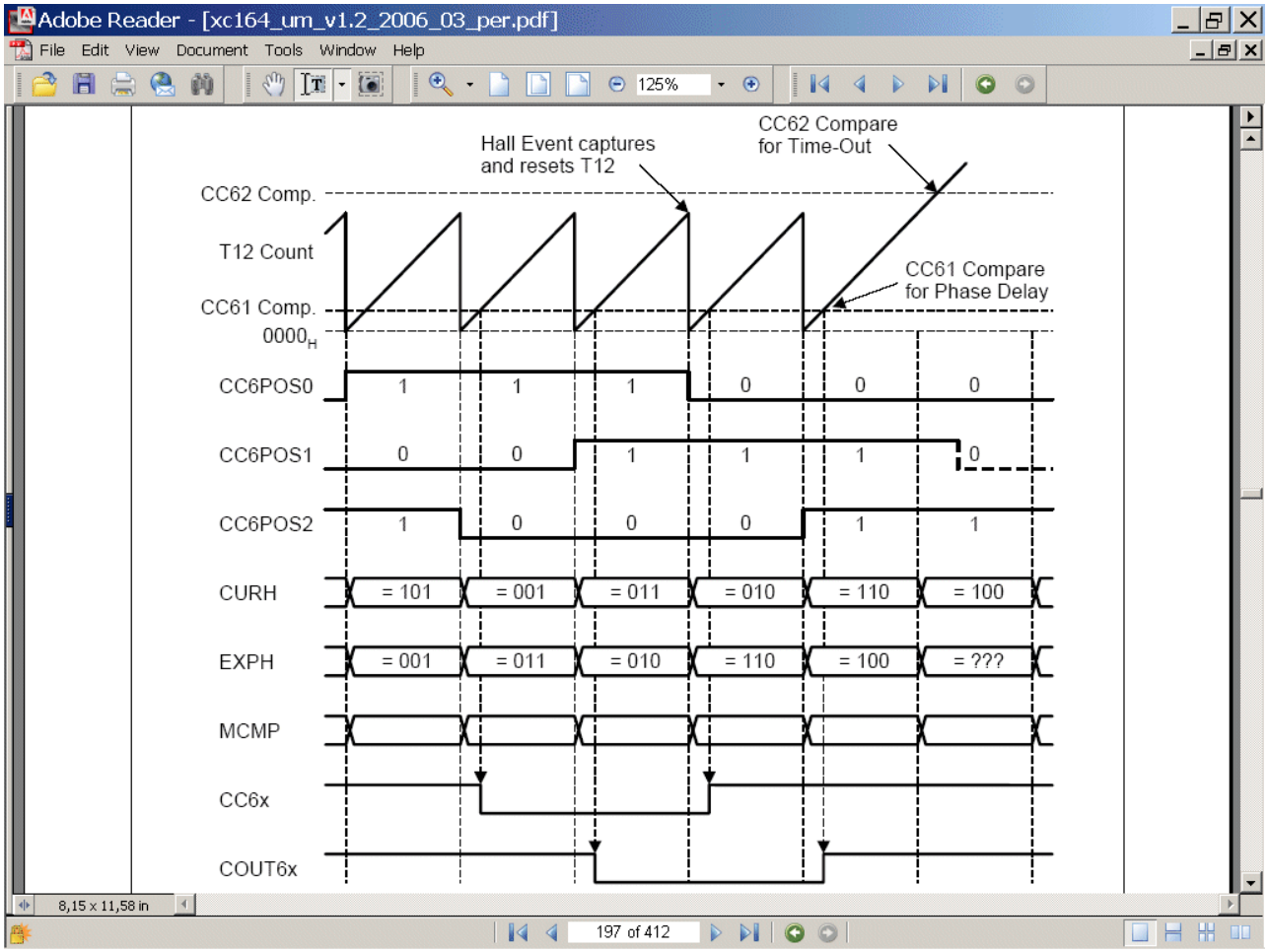


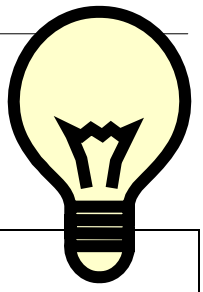
CAPCOM6_Timer_12 in Hall Sensor Mode:



actual motor speed

Brushless DC-Motor Control Example (all MSEL6x = 1000B):





Brushless DC-Motor Control with Timer T12 Block:

The CAPCOM6 provides a mode for the Timer T12 Block especially targeted for convenient control of Brushless DC-Motors.
In this mode, channel 0 is placed in capture mode, while channels 1 and 2 are in compare mode.

The signal to transfer the new compare values from the shadow registers (CC6xSR) into the actual compare registers (CC6xR) is now taken from the Correct Hall Event Compare.
In addition, this signal triggers a capture of the current T12 contents into register CC60R, and then forces a reset of T12 to 0000H.
The same signal is also used to perform the shadow transfer of the new T12 period value.

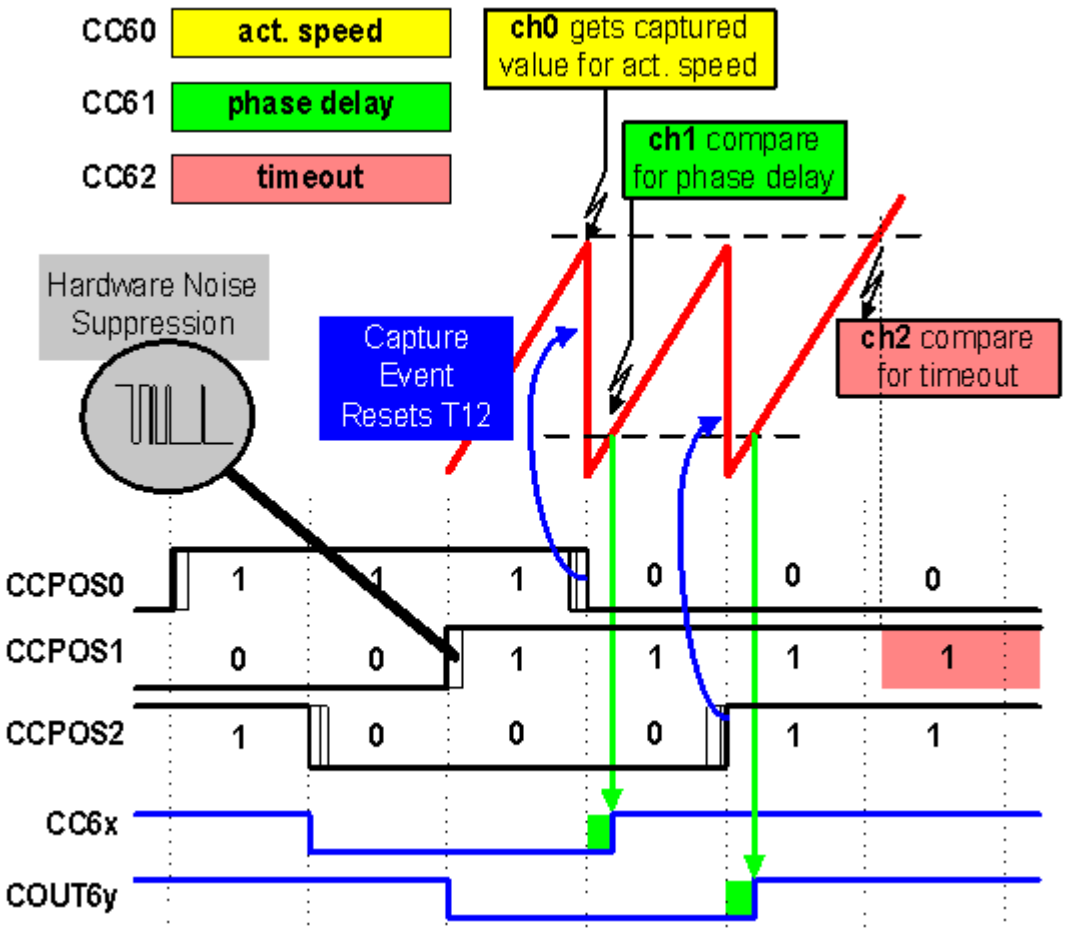
After the detection of a valid expected Hall pattern, the T12 count value is captured into channel 0 (representing the actual motor speed), and T12 is reset.
When the timer reaches the compare value in channel 1, the next multi-channel state is switched by triggering the shadow transfer of bitfield MCMP (if enabled in bitfield SWEN). This trigger event can be combined with several conditions which are necessary to implement a noise filtering (correct Hall event) and to synchronize the next multi-channel state to the modulation sources (avoiding spikes on the output lines).
This compare function of channel 1 can be used as a phase delay from the position sensor input signals to the switching of the output signals, which is necessary if a sensorless back-EMF technique is used instead of Hall sensors.
The compare value in channel 2 can be used as a timeout trigger (interrupt), indicating that the motor's actual speed is far below the desired destination value, which can be caused by an abnormal load change. In this mode, the modulation of the outputs by T12 needs to be disabled (T12MODENx = 0).

The capturing of the timer value in register CC60R, the shadow transfer from registers CC61SR to CC61R, from CC62SR to CC62R, and for the T12 period value, is done together with the reset event for T12.

Calculating the Speed of the Motor:

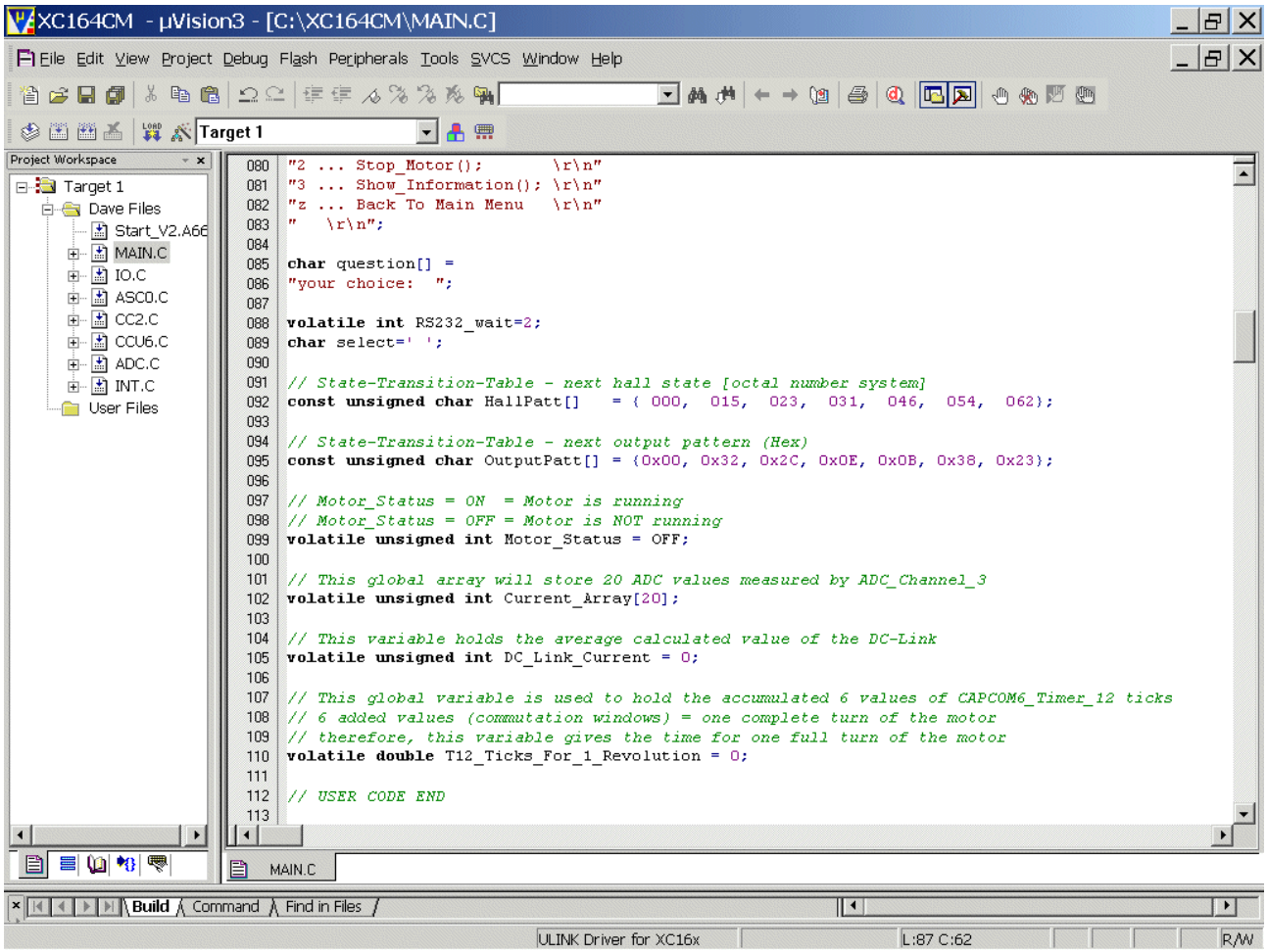
The CAPCOM6 Compare Channel 0 is used to capture the actual speed in Block Commutation Mode of one 60° window to register CC60R.

6 added values give the time for one full turn of the motor.



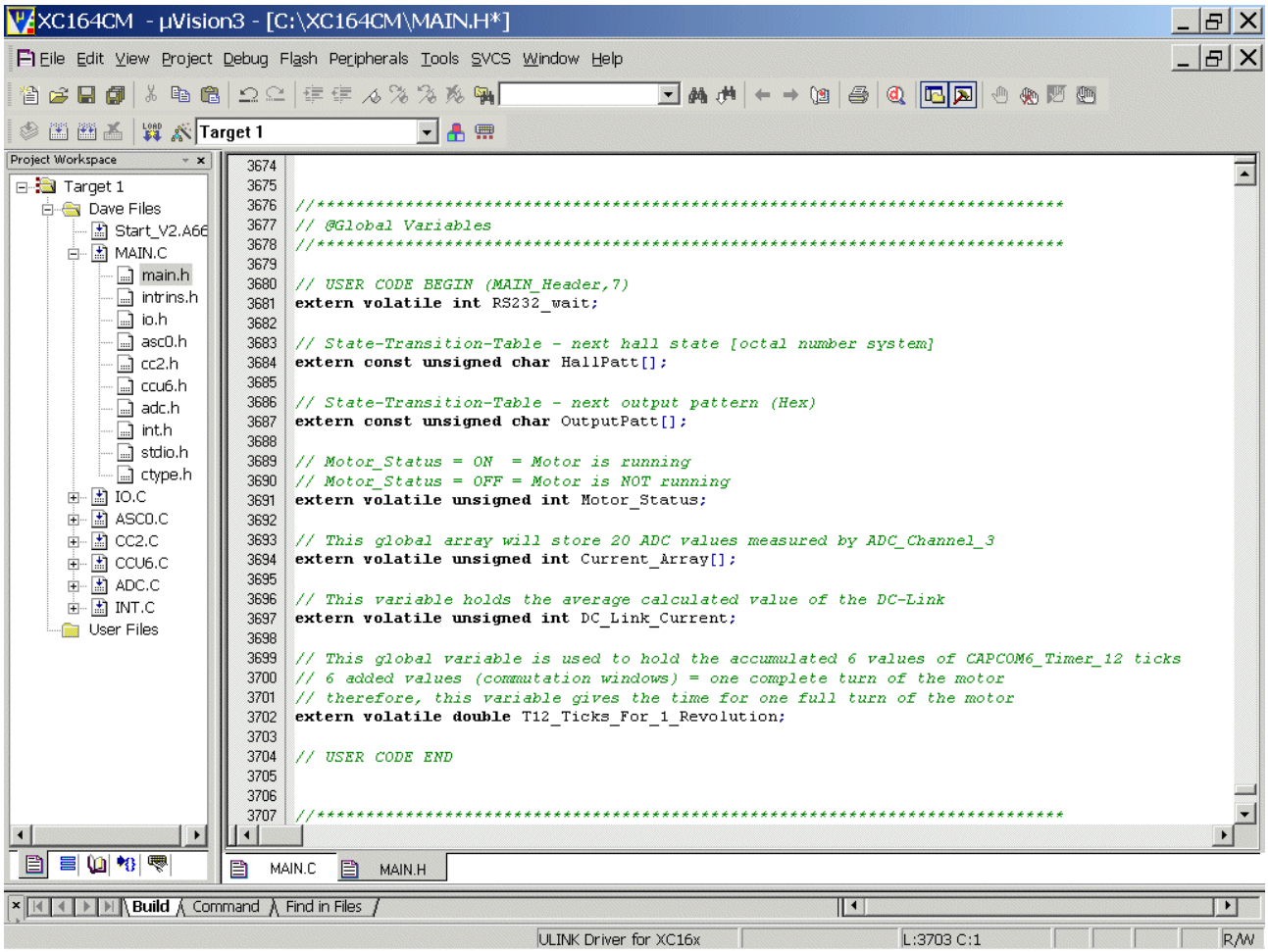
Double click MAIN.C and insert Global Variable:

```
// This global variable is used to hold the accumulated 6 values
of CAPCOM6_Timer_12 ticks
// 6 added values (commutation windows) = one complete turn of the
motor
// therefore, this variable gives the time for one full turn of
the motor
volatile double T12 Ticks For 1 Revolution = 0;
```



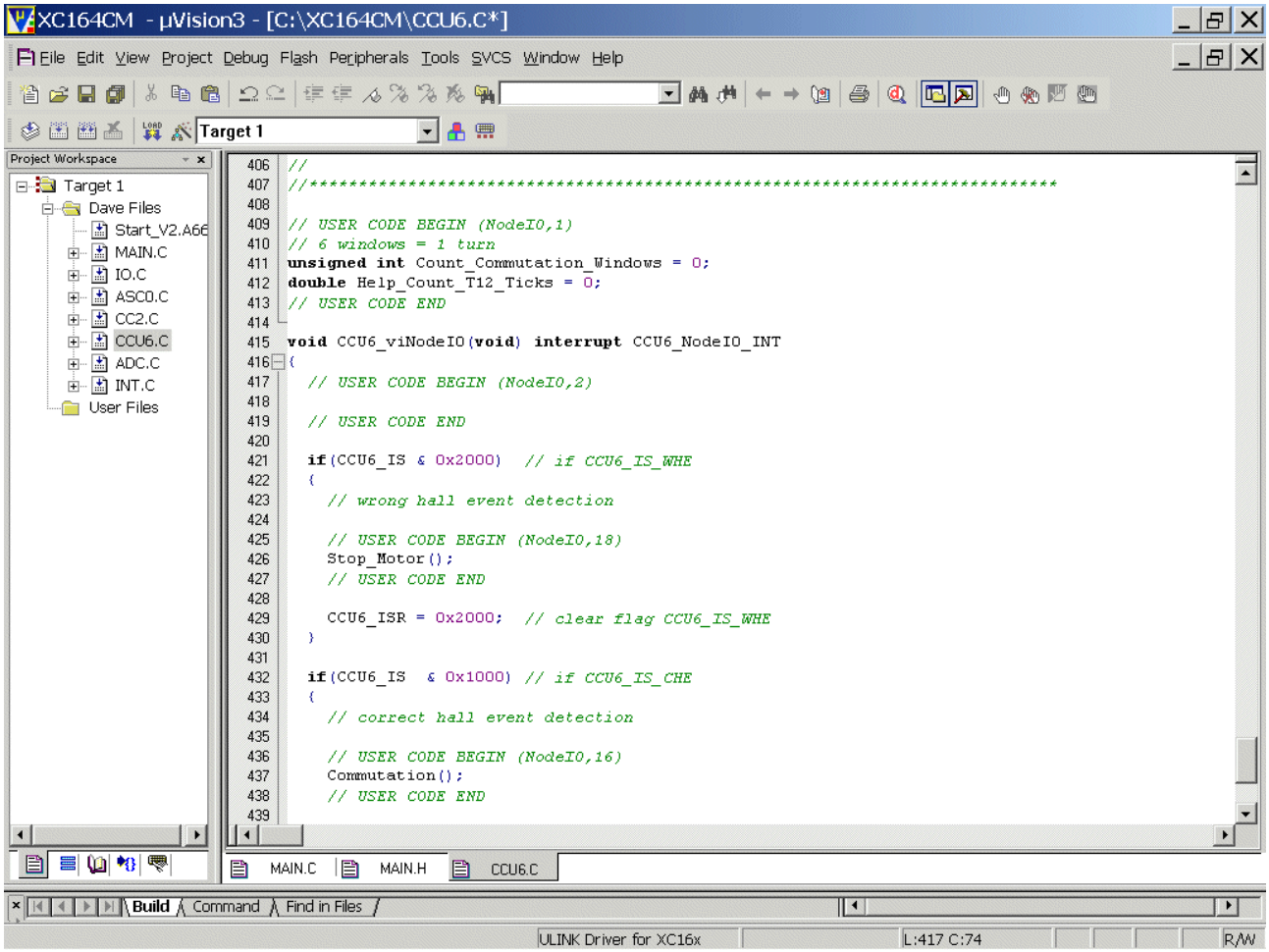
Double click main.h and insert Extern Declaration of Global Variable:

```
// This global variable is used to hold the accumulated 6 values
of CAPCOM6_Timer_12 ticks
// 6 added values (commutation windows) = one complete turn of the
motor
// therefore, this variable gives the time for one full turn of
the motor
extern volatile double T12 Ticks For 1 Revolution;
```



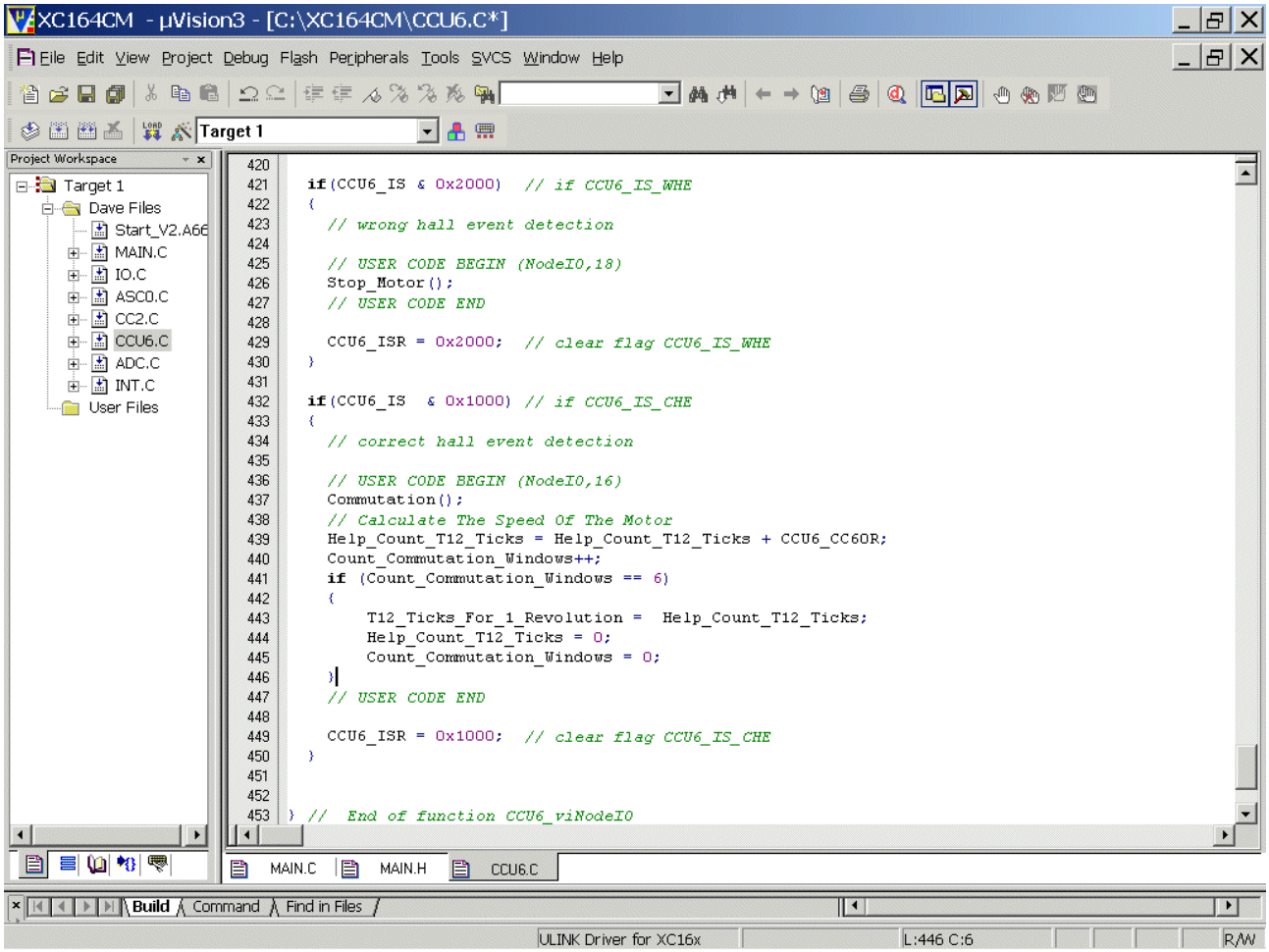
Double click CCU6.C and insert Modul Global Variables:

```
// 6 windows = 1 turn
unsigned int Count_Commutation_Windows = 0;
double Help Count T12 Ticks = 0;
```

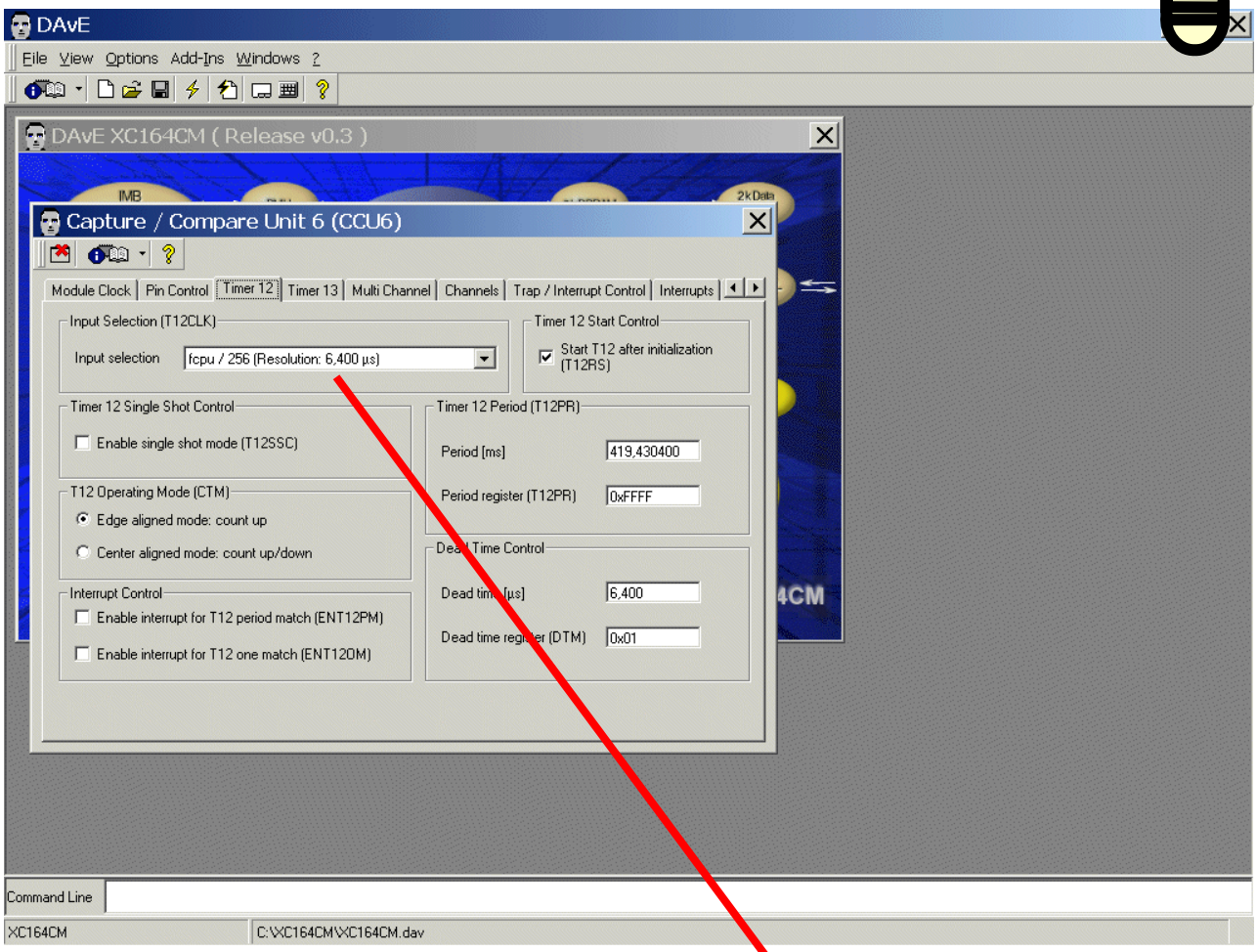


Double click CCU6.C and insert Code:

```
    // Calculate The Speed Of The Motor
    Help_Count_T12_Ticks = Help_Count_T12_Ticks + CCU6_CC60R;
    Count_Commutation_Windows++;
    if (Count_Commutation_Windows == 6)
    {
        T12_Ticks_For_1_Revolution =  Help_Count_T12_Ticks;
        Help_Count_T12_Ticks = 0;
        Count_Commutation_Windows = 0;
    }
```



Information about the Resolution of CAPCOM6_Timer 12:



Therefore:

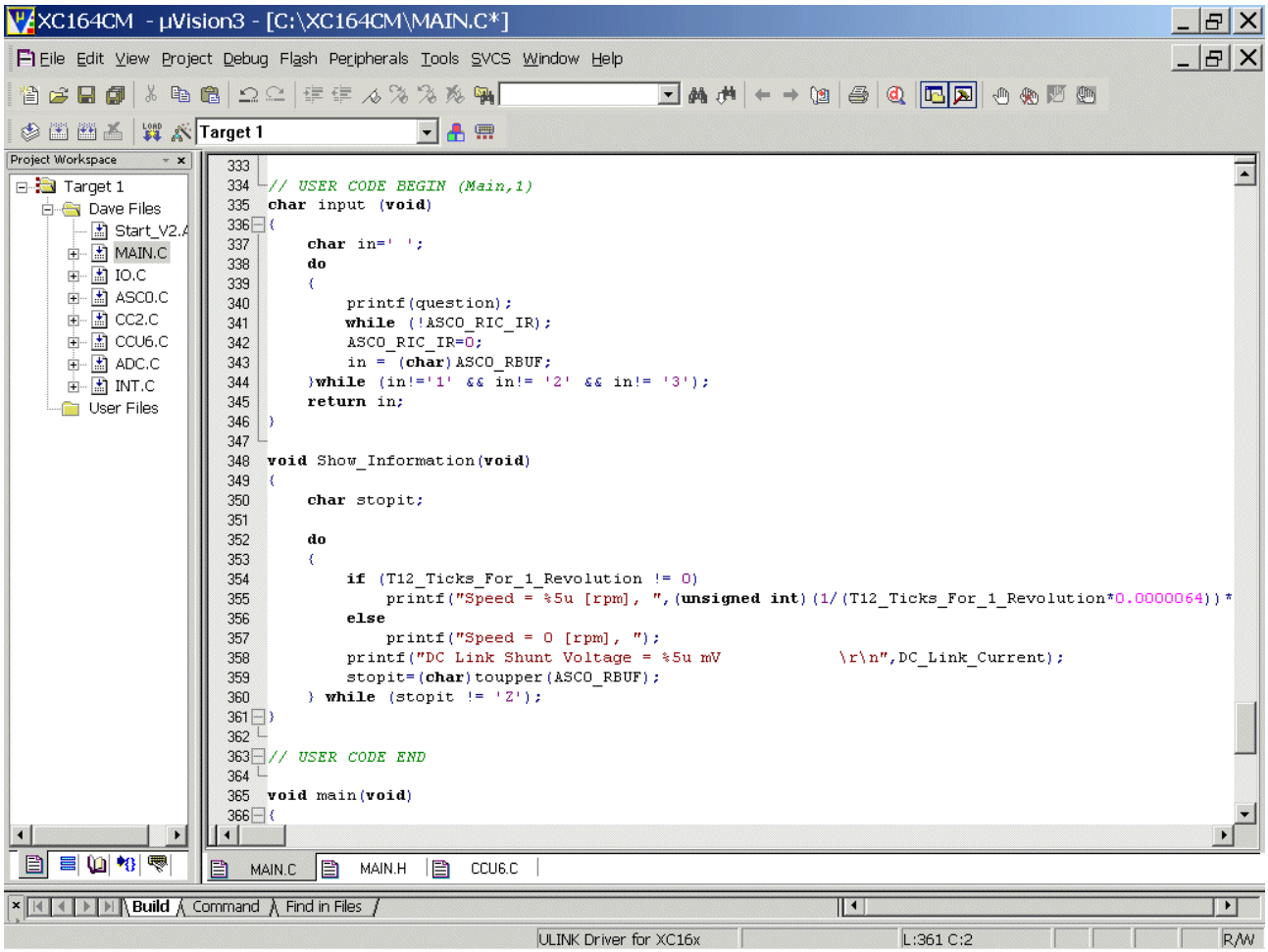
revolutions-per-minute [rpm] = (1/(T12_Ticks_For_1_Revolution * ($6{,}4 * 10^{-6}$)) * 60

Double click MAIN.C and change Code from:

```
void Show_Information(void)
{
    char stopit;

    do
    {
        printf("DC Link Shunt Voltage = %5u mV                    \r",DC_Link_Current);
        stopit=(char)toupper(ASC0_RBUF);
    } while (stopit != 'Z');
}
```

to:

```
void Show_Information(void)
{
    char stopit;

    do
    {
        if (T12_Ticks_For_1_Revolution != 0)
            printf("Speed = %5u [rpm], ",(unsigned
int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*60);
        else
            printf("Speed = 0 [rpm], ");
        printf("DC Link Shunt Voltage = %5u mV
\r\n",DC_Link_Current);
        stopit=(char)toupper(ASC0_RBUF);
    } while (stopit != 'Z');
}
```

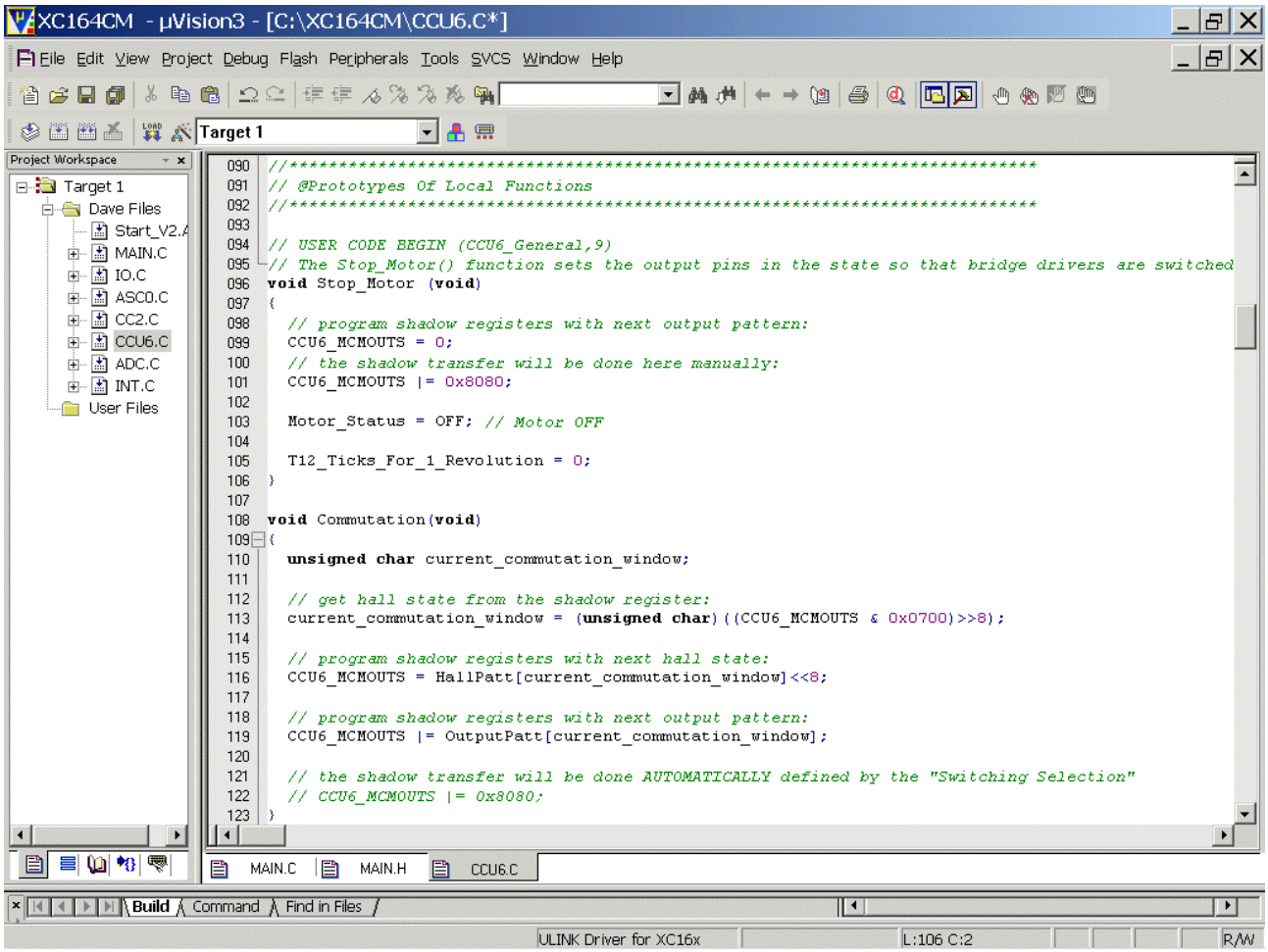

XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
333
334 // USER CODE BEGIN (Main,1)
335 char input (void)
336 {
337 char in=' ';
338 do
339 {
340 printf(question);
341 while (!ASC0_RIC_IR);
342 ASC0_RIC_IR=0;
343 in = (char)ASC0_RBUF;
344 }while (in!='1' && in!= '2' && in!= '3');
345 return in;
346 }
347
348 void Show_Information(void)
349 {
350 char stopit;
351
352 do
353 {
354 if (T12_Ticks_For_1_Revolution != 0)
355 printf("Speed = %5u [rpm], ",(unsigned int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*
356 else
357 printf("Speed = 0 [rpm], ");
358 printf("DC Link Shunt Voltage = %5u mV \r\n",DC_Link_Current);
359 stopit=(char)toupper(ASC0_RBUF);
360 } while (stopit != 'Z');
361 }
362
363 // USER CODE END
364
365 void main(void)
366 {
MAIN.C
MAIN.H
CCU6.C
Build
Command
Find in Files
ULINK Driver for XC16x
L:361 C:2
R/W

Double click CCU6.C and change Code from:

```
// The Stop_Motor() function sets the output pins in the state so
that bridge drivers are switched off
void Stop_Motor (void)
{
  // program shadow registers with next output pattern:
  CCU6_MCMOUTS = 0;
  // the shadow transfer will be done here manually:
  CCU6_MCMOUTS |= 0x8080;

  Motor_Status = OFF; // Motor OFF
}
```

to:

```
// The Stop_Motor() function sets the output pins in the state so
that bridge drivers are switched off
void Stop_Motor (void)
{
  // program shadow registers with next output pattern:
  CCU6_MCMOUTS = 0;
  // the shadow transfer will be done here manually:
  CCU6_MCMOUTS |= 0x8080;

  Motor_Status = OFF; // Motor OFF

  T12_Ticks_For_1_Revolution = 0;
}
```

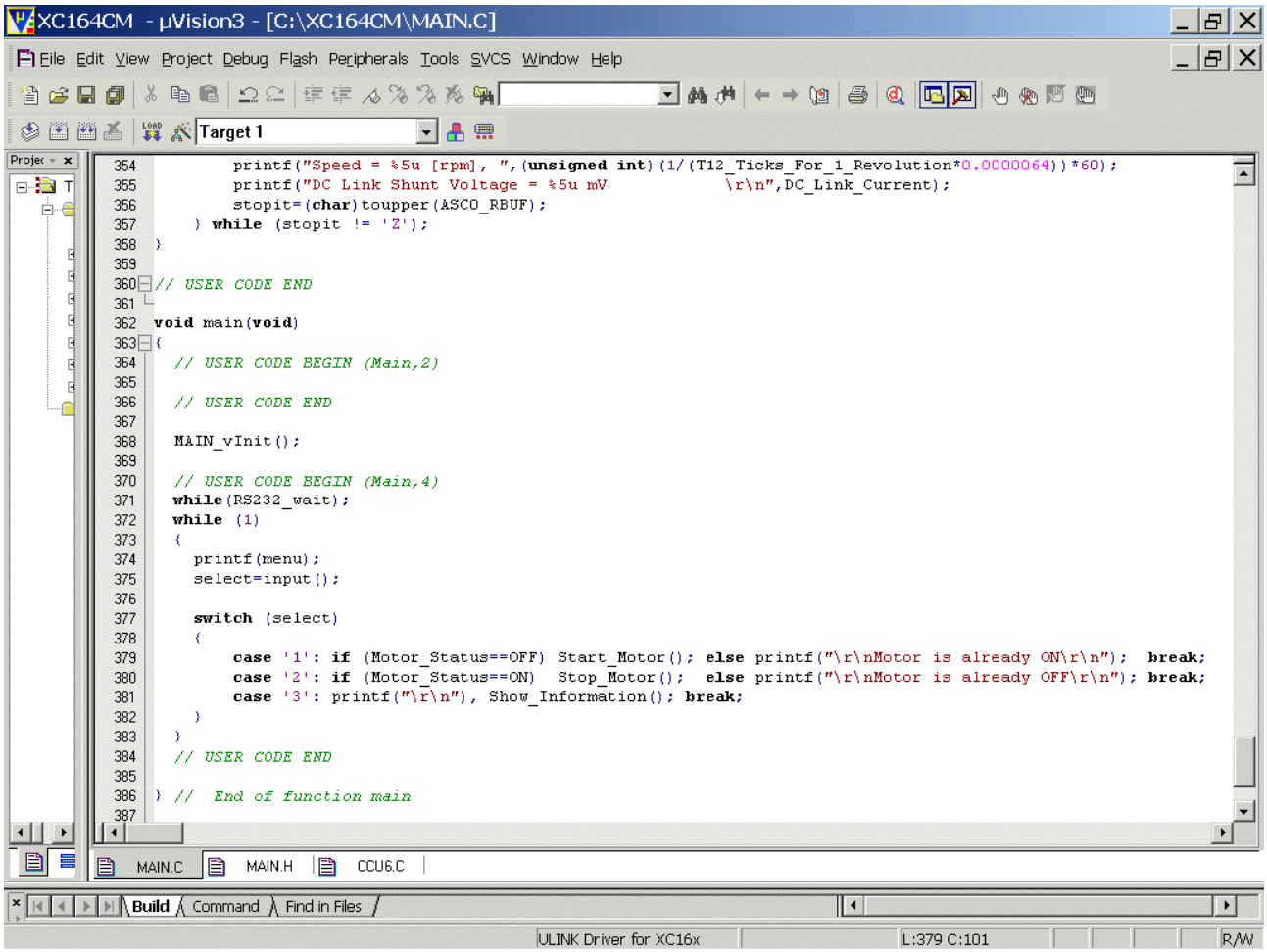

XC164CM - µVision3 - [C:\XC164CM\CCU6.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
090 //***
091 // @Prototypes Of Local Functions
092 //***
093
094 // USER CODE BEGIN (CCU6_General,9)
095 // The Stop_Motor() function sets the output pins in the state so that bridge drivers are switched
096 void Stop_Motor (void)
097 {
098 // program shadow registers with next output pattern:
099 CCU6_MCMOUTS = 0;
100 // the shadow transfer will be done here manually:
101 CCU6_MCMOUTS |= 0x8080;
102
103 Motor_Status = OFF; // Motor OFF
104
105 T12_Ticks_For_1_Revolution = 0;
106 }
107
108 void Commutation(void)
109 {
110 unsigned char current_commutation_window;
111
112 // get hall state from the shadow register:
113 current_commutation_window = (unsigned char)((CCU6_MCMOUTS & 0x0700)>>8);
114
115 // program shadow registers with next hall state:
116 CCU6_MCMOUTS = HallPatt[current_commutation_window]<<8;
117
118 // program shadow registers with next output pattern:
119 CCU6_MCMOUTS |= OutputPatt[current_commutation_window];
120
121 // the shadow transfer will be done AUTOMATICALLY defined by the "Switching Selection"
122 // CCU6_MCMOUTS |= 0x8080;
123 }
MAIN.C
MAIN.H
CCU6.C
Build
Command
Find in Files
ULINK Driver for XC16x
L:106 C:2
R/W

Double click MAIN.C and change Code from:

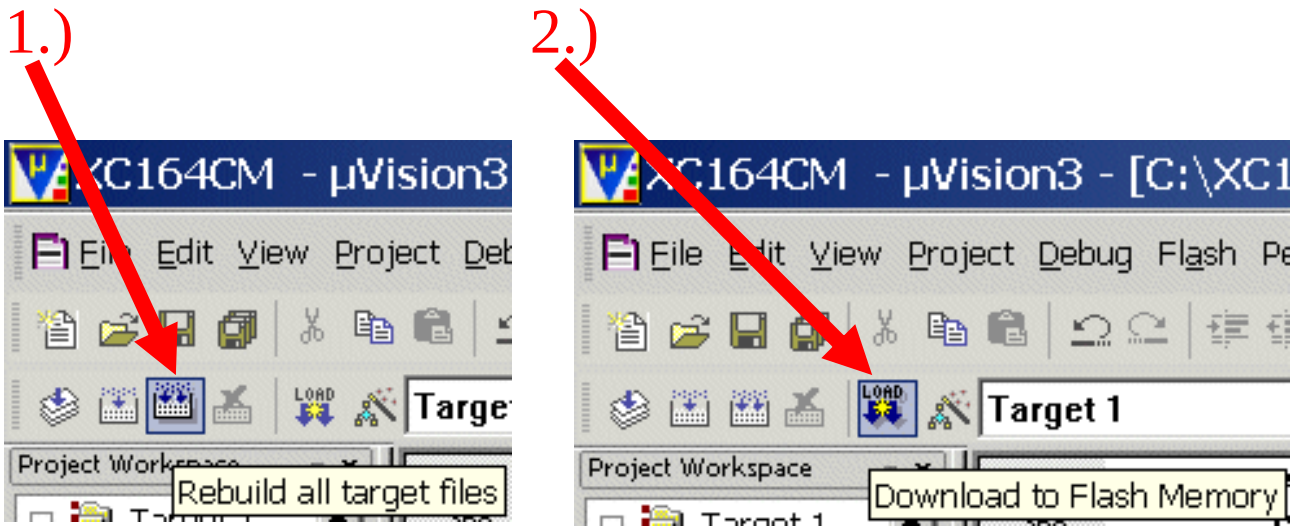
```
    switch (select)
    {
        case '1': Start_Motor(); break;
        case '2': Stop_Motor();  break;
        case '3': printf("\r\n"), Show_Information(); break;
    }
```

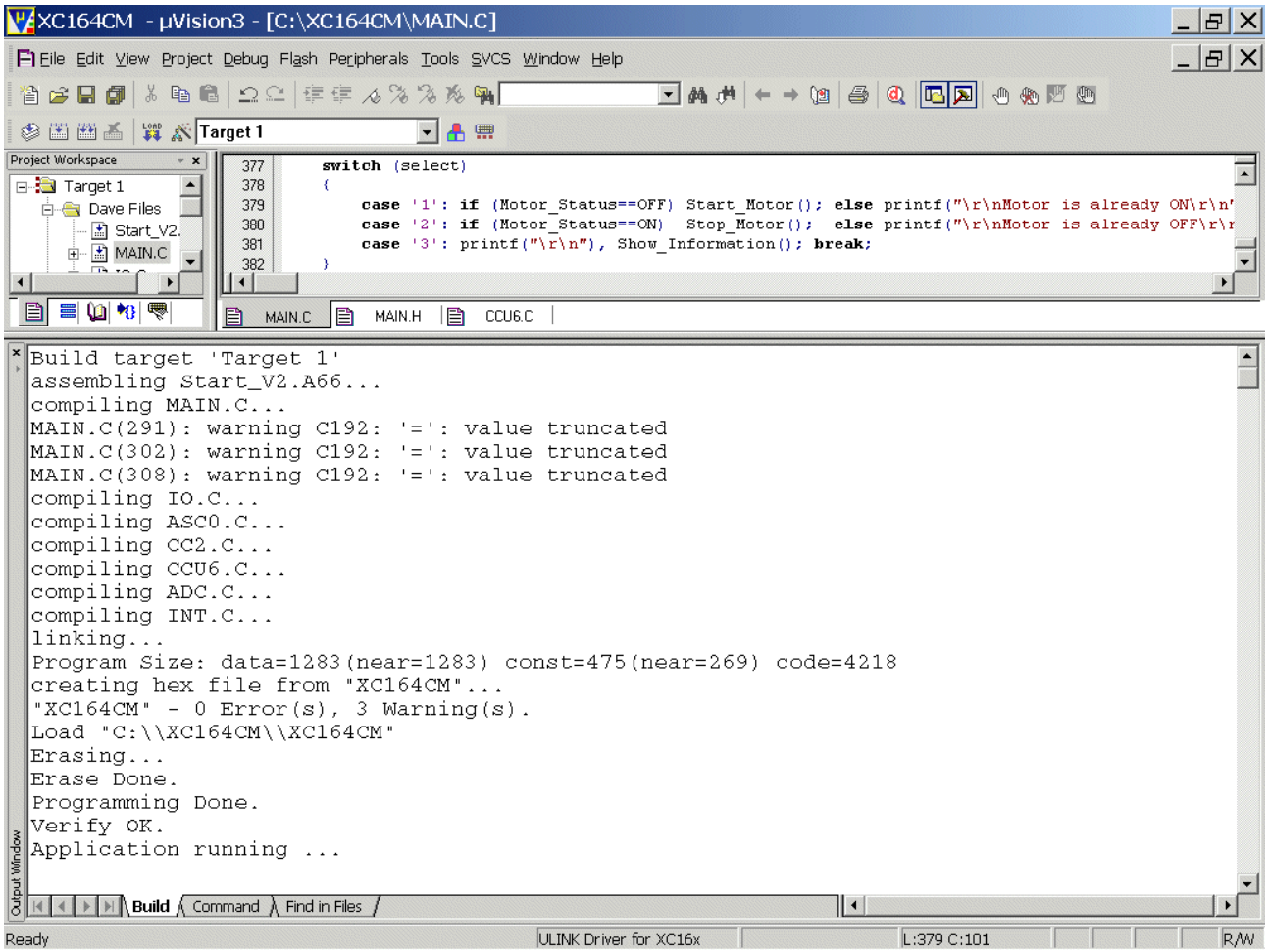
to:

```
    switch (select)
    {
        case '1': if (Motor_Status==OFF) Start_Motor(); else
printf("\r\nMotor is already ON\r\n");  break;
        case '2': if (Motor_Status==ON)  Stop_Motor();  else
printf("\r\nMotor is already OFF\r\n"); break;
        case '3': printf("\r\n"), Show_Information(); break;
    }
```

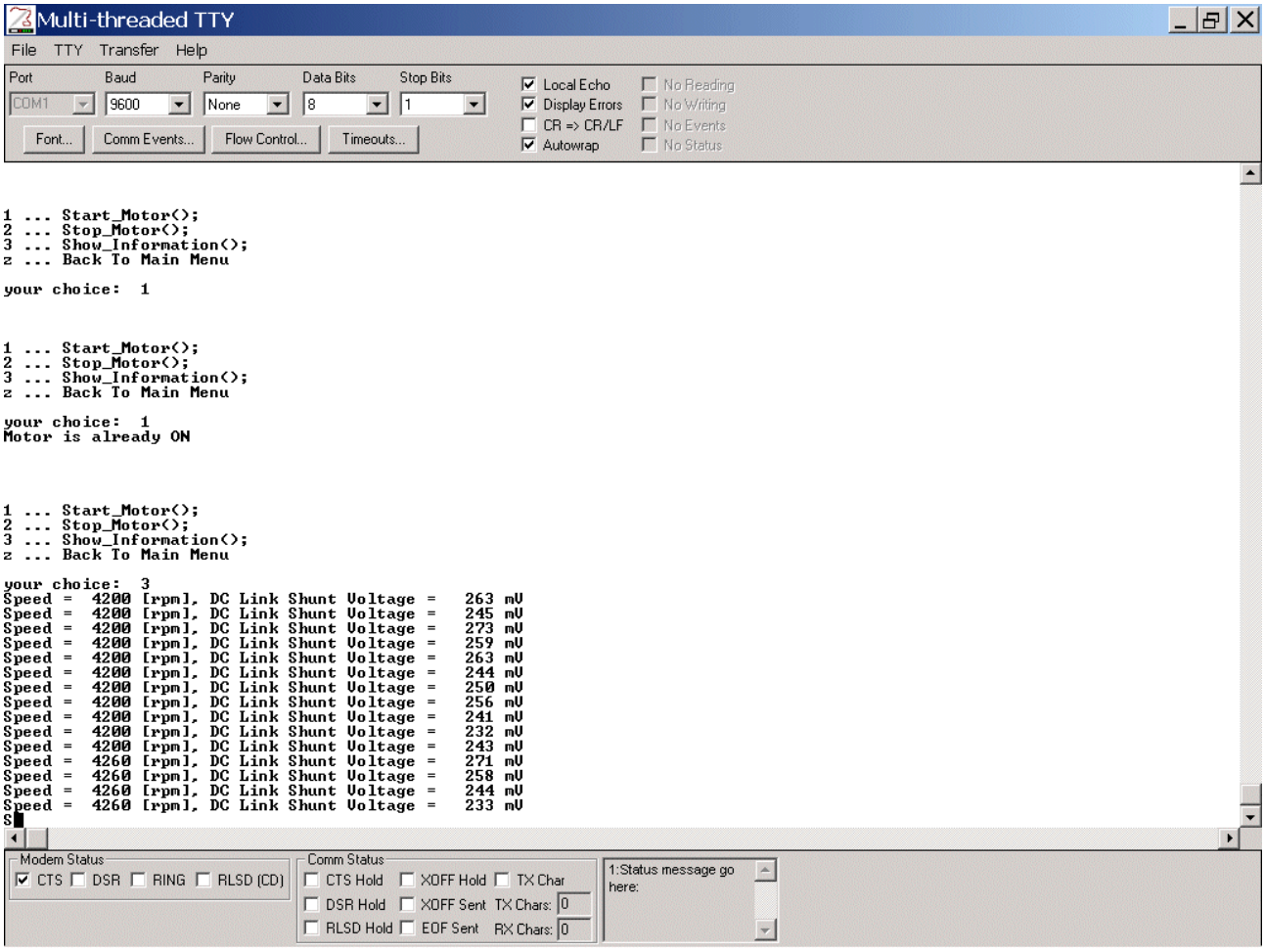


Build Application:





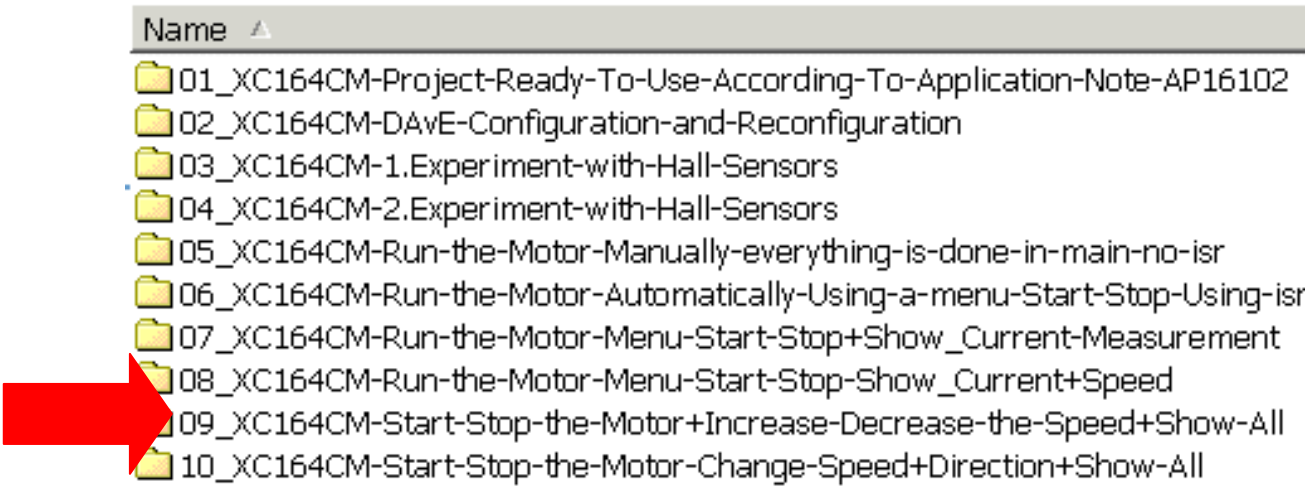
Run Program:


Multi-threaded TTY
File TTY Transfer Help
Port Baud Parity Data Bits Stop Bits
COM1 9600 None 8 1
Font... Comm Events... Flow Control... Timeouts...
Local Echo
Display Errors
CR => CR/LF
Autowrap
No Reading
No Writing
No Events
No Status
1 ... Start_Motor();
2 ... Stop_Motor();
3 ... Show_Information();
z ... Back To Main Menu
your choice: 1
1 ... Start_Motor();
2 ... Stop_Motor();
3 ... Show_Information();
z ... Back To Main Menu
your choice: 1
Motor is already ON
1 ... Start_Motor();
2 ... Stop_Motor();
3 ... Show_Information();
z ... Back To Main Menu
your choice: 3
Speed = 4200 [rpm], DC Link Shunt Voltage = 263 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 245 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 273 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 259 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 263 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 244 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 250 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 256 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 241 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 232 mV
Speed = 4200 [rpm], DC Link Shunt Voltage = 243 mV
Speed = 4260 [rpm], DC Link Shunt Voltage = 271 mV
Speed = 4260 [rpm], DC Link Shunt Voltage = 258 mV
Speed = 4260 [rpm], DC Link Shunt Voltage = 244 mV
Speed = 4260 [rpm], DC Link Shunt Voltage = 233 mV
S
Modem Status
CTS DSR RING RLSD (CD)
Comm Status
CTS Hold XOFF Hold TX Char
DSR Hold XOFF Sent TX Chars: 0
RLSD Hold EOF Sent RX Chars: 0
1:Status message go here:

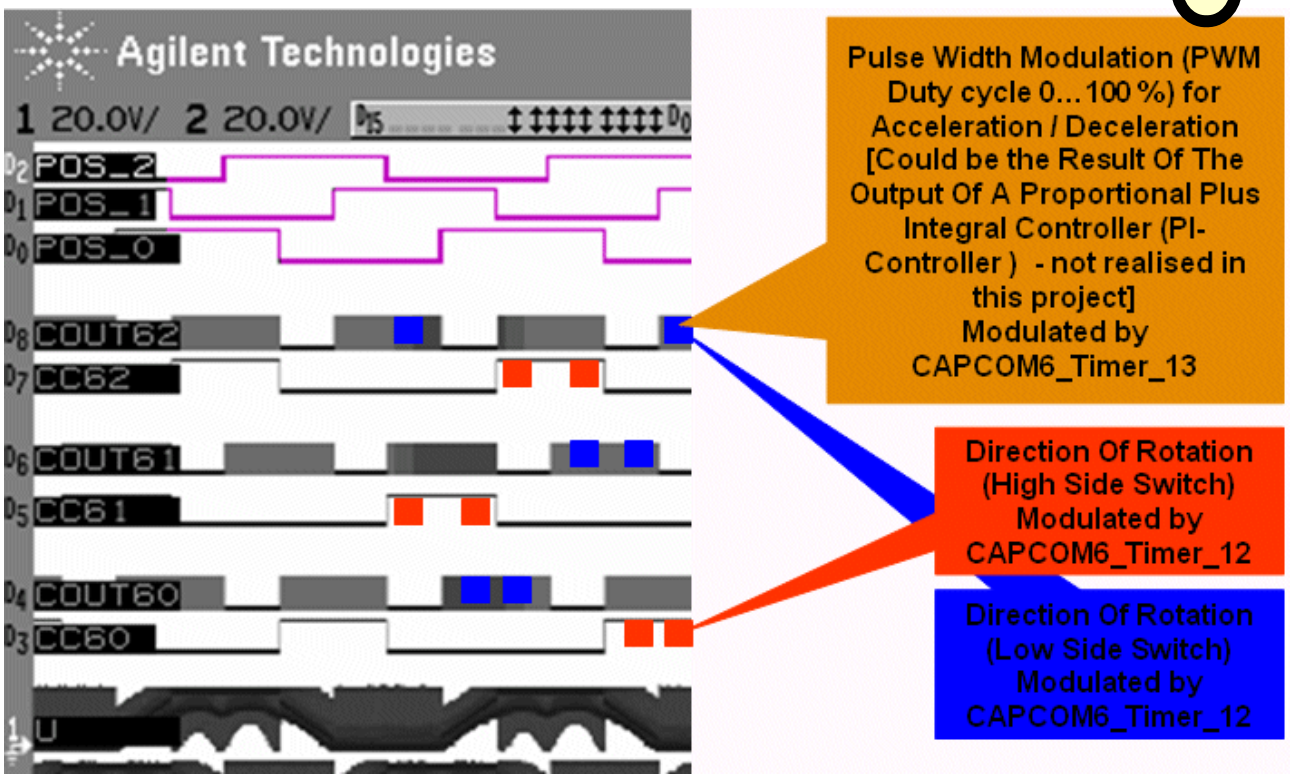
9.)

Changing The Speed Of The Motor:

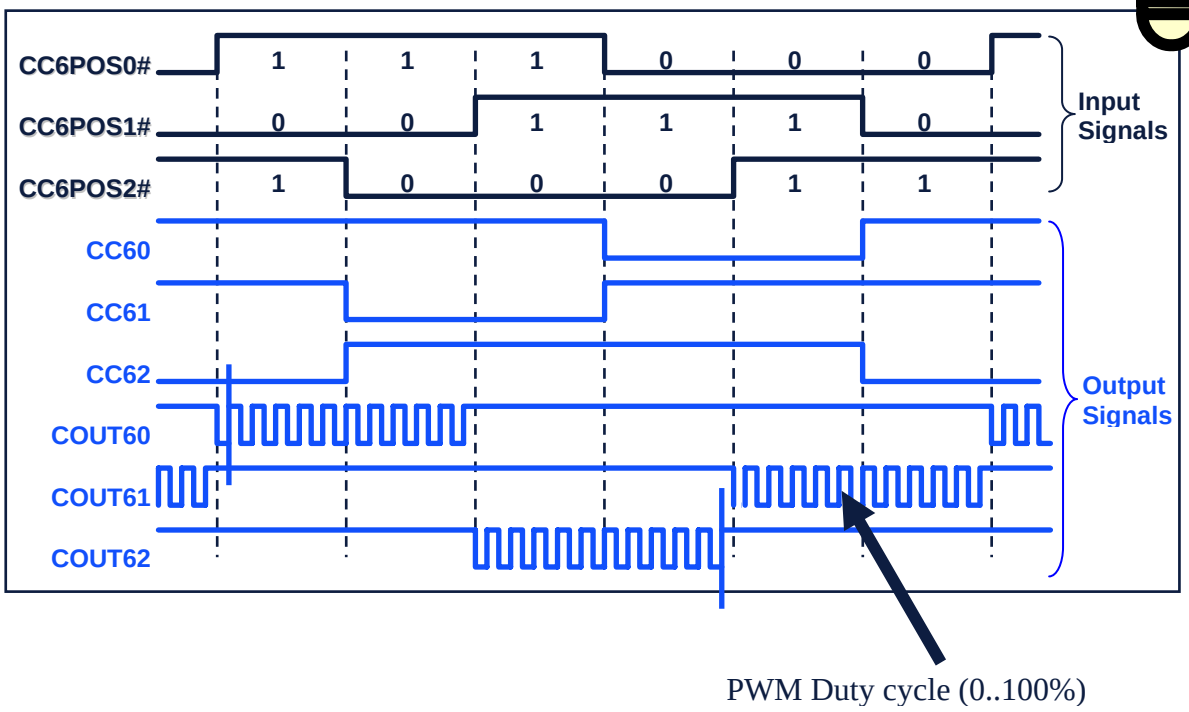
Relevant Project:



Register Information / Additional Information:



Register Information / Additional Information:



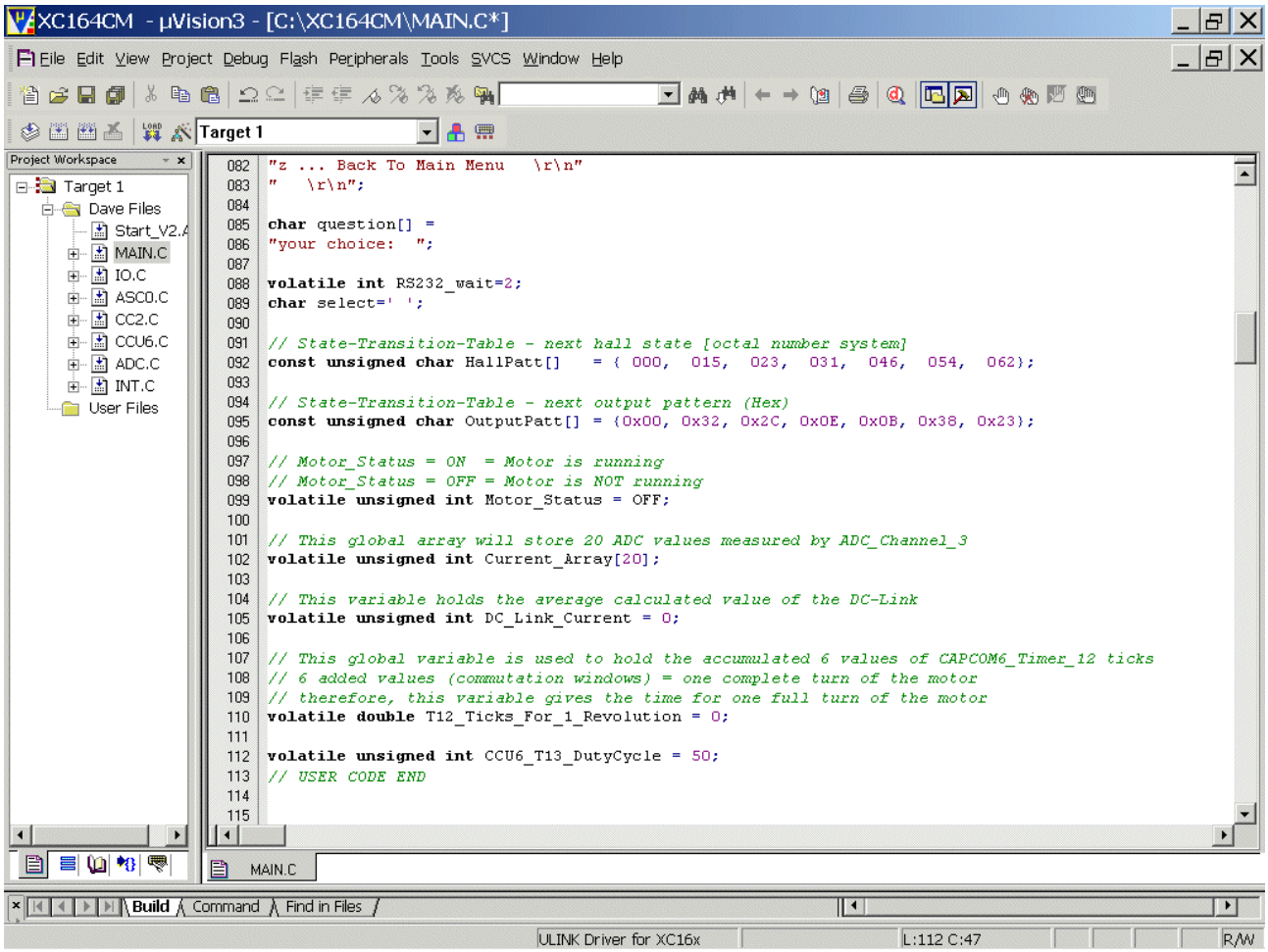
PWM Duty cycle (0..100%)

To change the speed of the motor we will add the following functions:
void CCU6_T13_UpdateDutyCycle(void) { … }
void CCU6_T13_IncreaseDutyCycle(void) { … }
void CCU6_T13_DecreaseDutyCycle(void) { … }

To control the motor speed we are going to add the following function-calls
CCU6_T13_IncreaseDutyCycle();
CCU6_T13_DecreaseDutyCycle();
into the main menu.

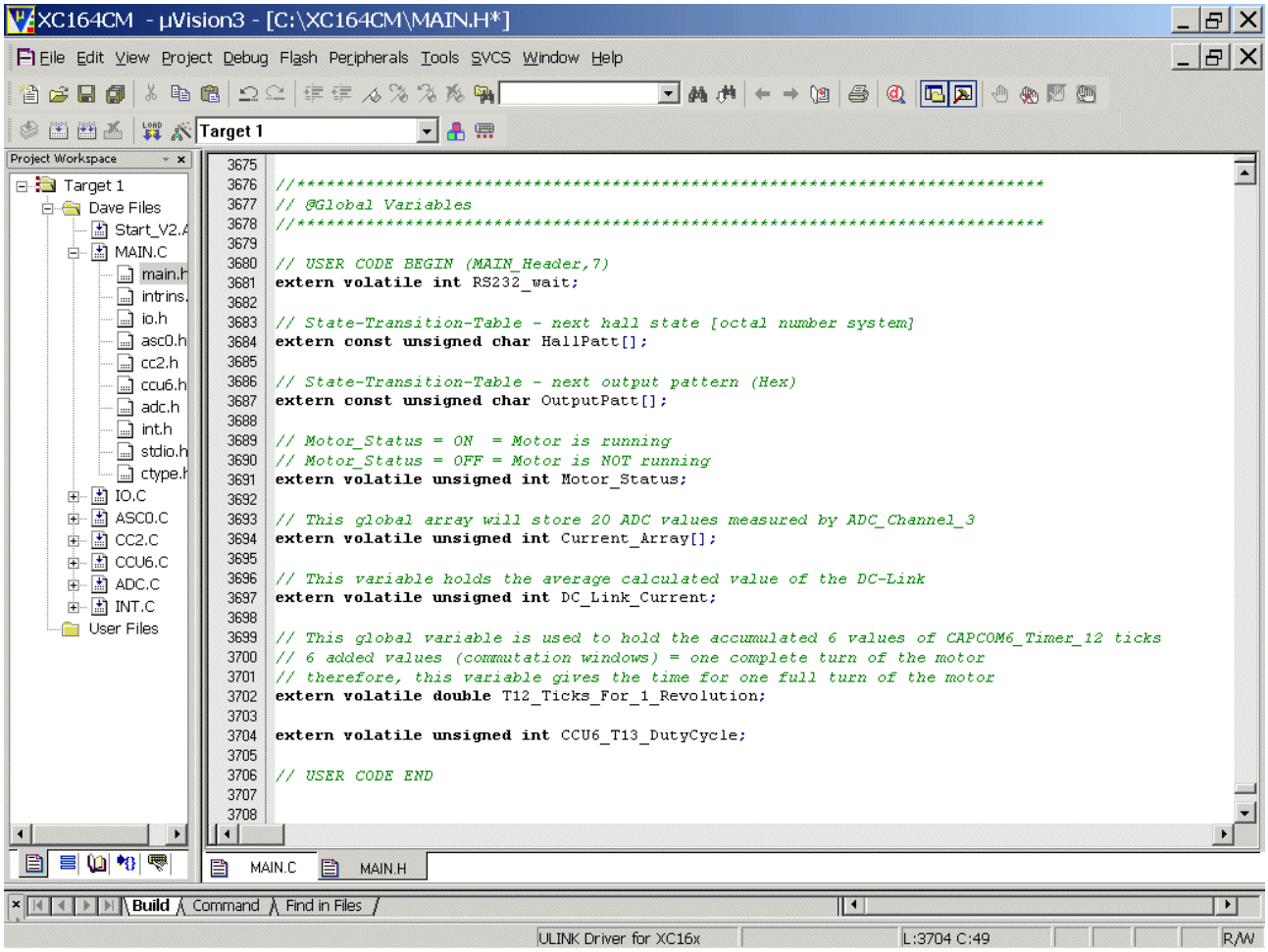
Double click MAIN.C and insert Global Variable:

```
volatile unsigned int CCU6_T13_DutyCycle = 50;
```



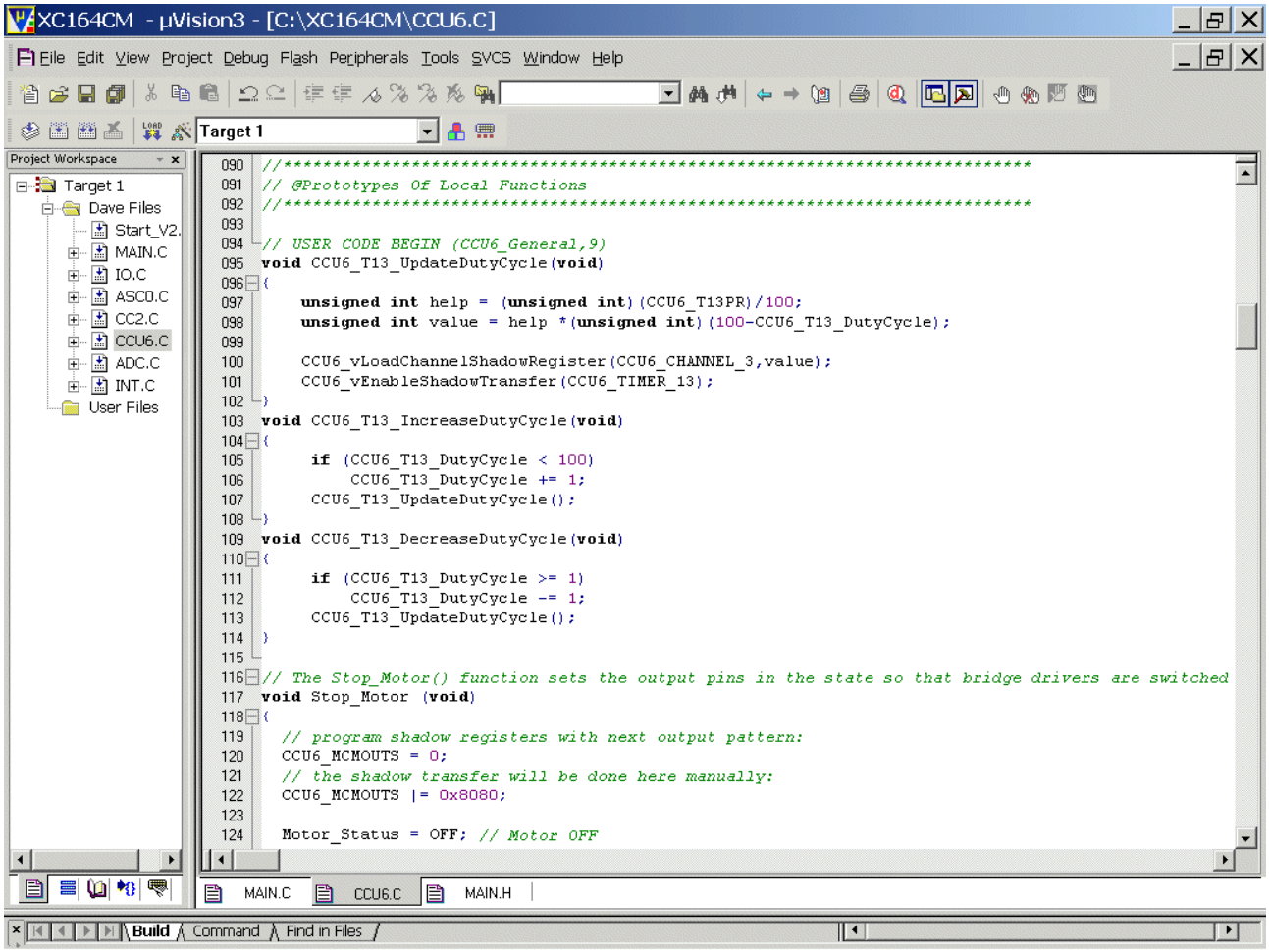
Double click main.h and insert Extern Declariation of Global Variable:

```
extern volatile unsigned int CCU6_T13_DutyCycle;
```



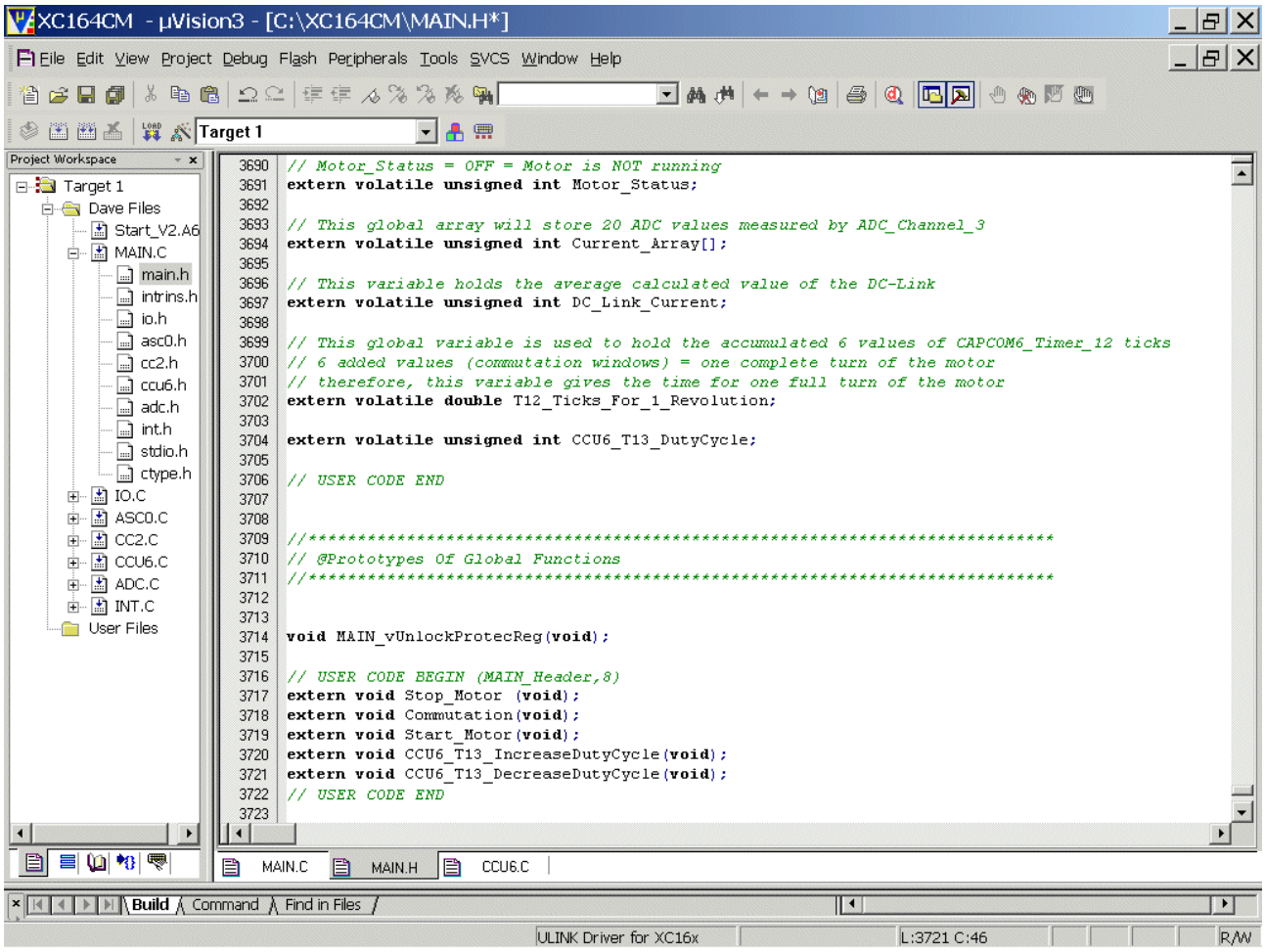
Double click CCU6.C and insert Code:

```
void CCU6_T13_UpdateDutyCycle(void)
{
    unsigned int help = (unsigned int)(CCU6_T13PR)/100;
    unsigned int value = help *(unsigned int)(100-
CCU6_T13_DutyCycle);

    CCU6_vLoadChannelShadowRegister(CCU6_CHANNEL_3,value);
    CCU6_vEnableShadowTransfer(CCU6_TIMER_13);
}
void CCU6_T13_IncreaseDutyCycle(void)
{
     if (CCU6_T13_DutyCycle < 100)
         CCU6_T13_DutyCycle += 1;
     CCU6_T13_UpdateDutyCycle();
}
void CCU6_T13_DecreaseDutyCycle(void)
{
     if (CCU6_T13_DutyCycle >= 1)
         CCU6_T13_DutyCycle -= 1;
     CCU6_T13_UpdateDutyCycle();
}
```



Double click main.h and insert Extern Declarations of Prototypes of Global Function:

```
extern void CCU6_T13_IncreaseDutyCycle(void);
extern void CCU6_T13_DecreaseDutyCycle(void);
```

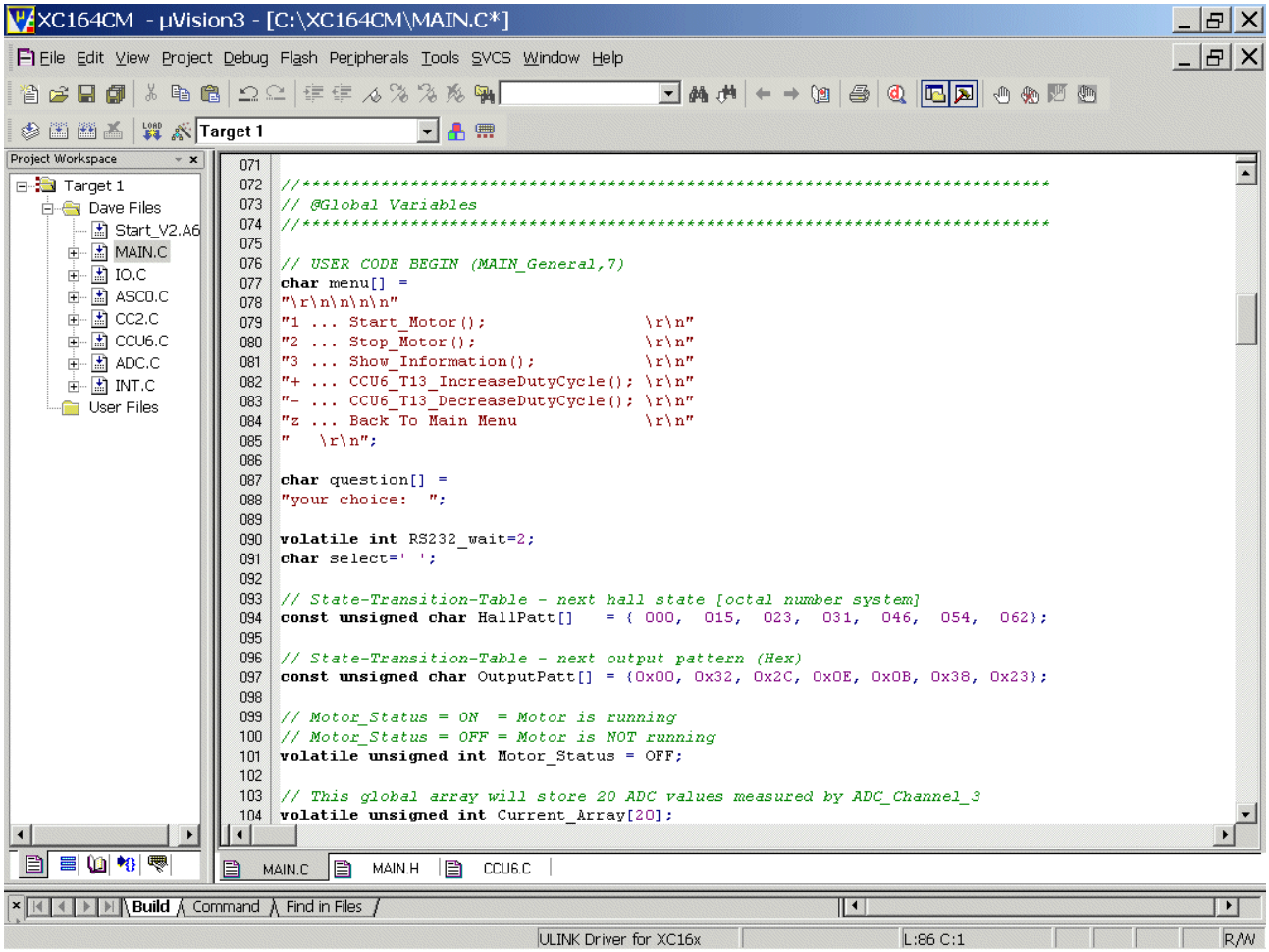


Double click MAIN.C and change Global Variable from:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor();       \r\n"
"2 ... Stop_Motor();        \r\n"
"3 ... Show_Information();  \r\n"
"z ... Back To Main Menu    \r\n"
"    \r\n";
```

to:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor();                  \r\n"
"2 ... Stop_Motor();                   \r\n"
"3 ... Show_Information();              \r\n"
"+ ... CCU6_T13_IncreaseDutyCycle();   \r\n"
"- ... CCU6_T13_DecreaseDutyCycle();   \r\n"
"z ... Back To Main Menu               \r\n"
"    \r\n";
```

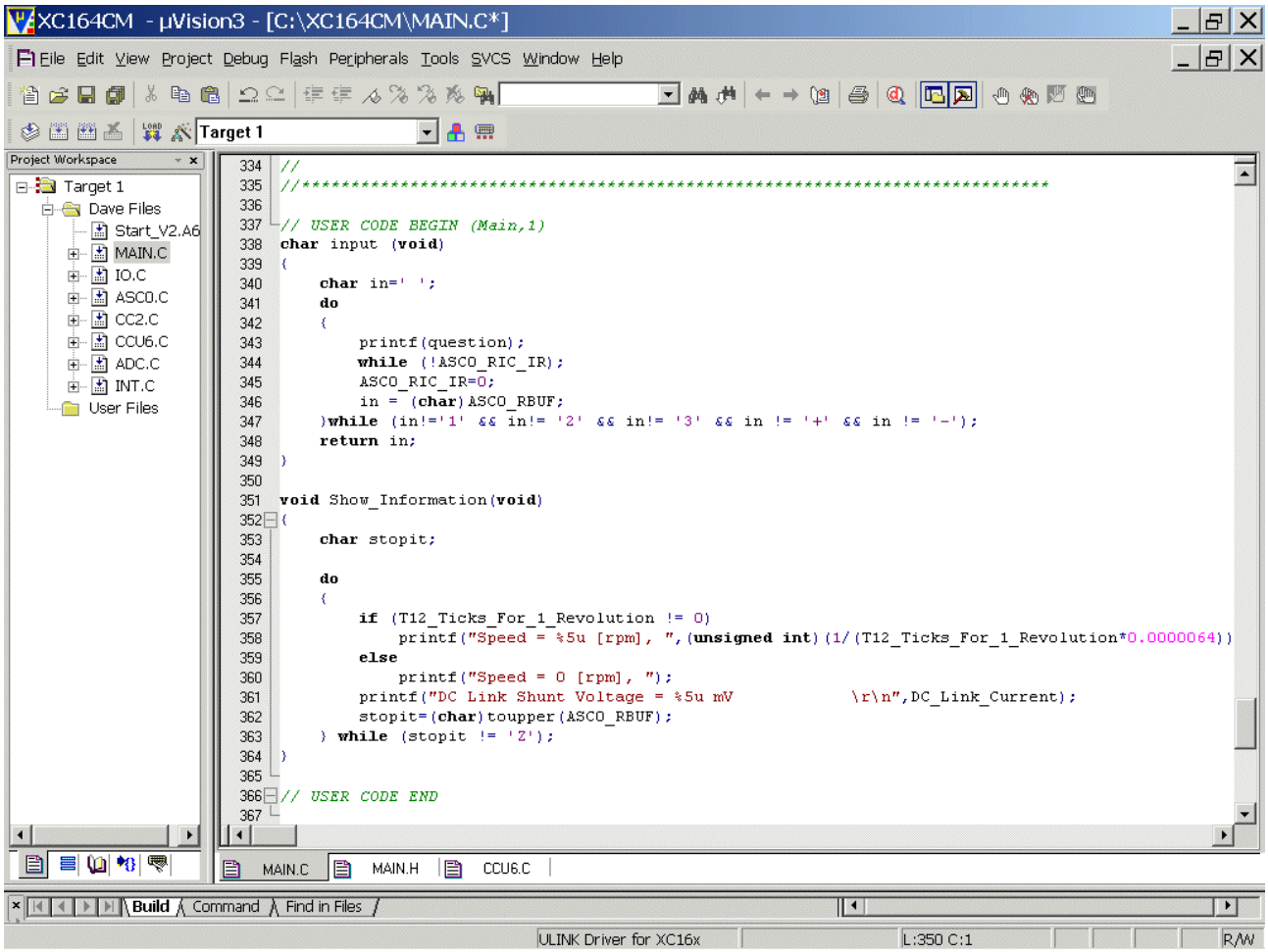


Double click MAIN.C and change Code from:

```
char input (void)
{
     char in=' ';
     do
     {
          printf(question);
          while (!ASC0_RIC_IR);
          ASC0_RIC_IR=0;
          in = (char)ASC0_RBUF;
     }while (in!='1' && in!= '2' && in!= '3');
     return in;
}
```

to:

```
char input (void)
{
     char in=' ';
     do
     {
          printf(question);
          while (!ASC0_RIC_IR);
          ASC0_RIC_IR=0;
          in = (char)ASC0_RBUF;
     }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in
!= '-');
     return in;
}
```

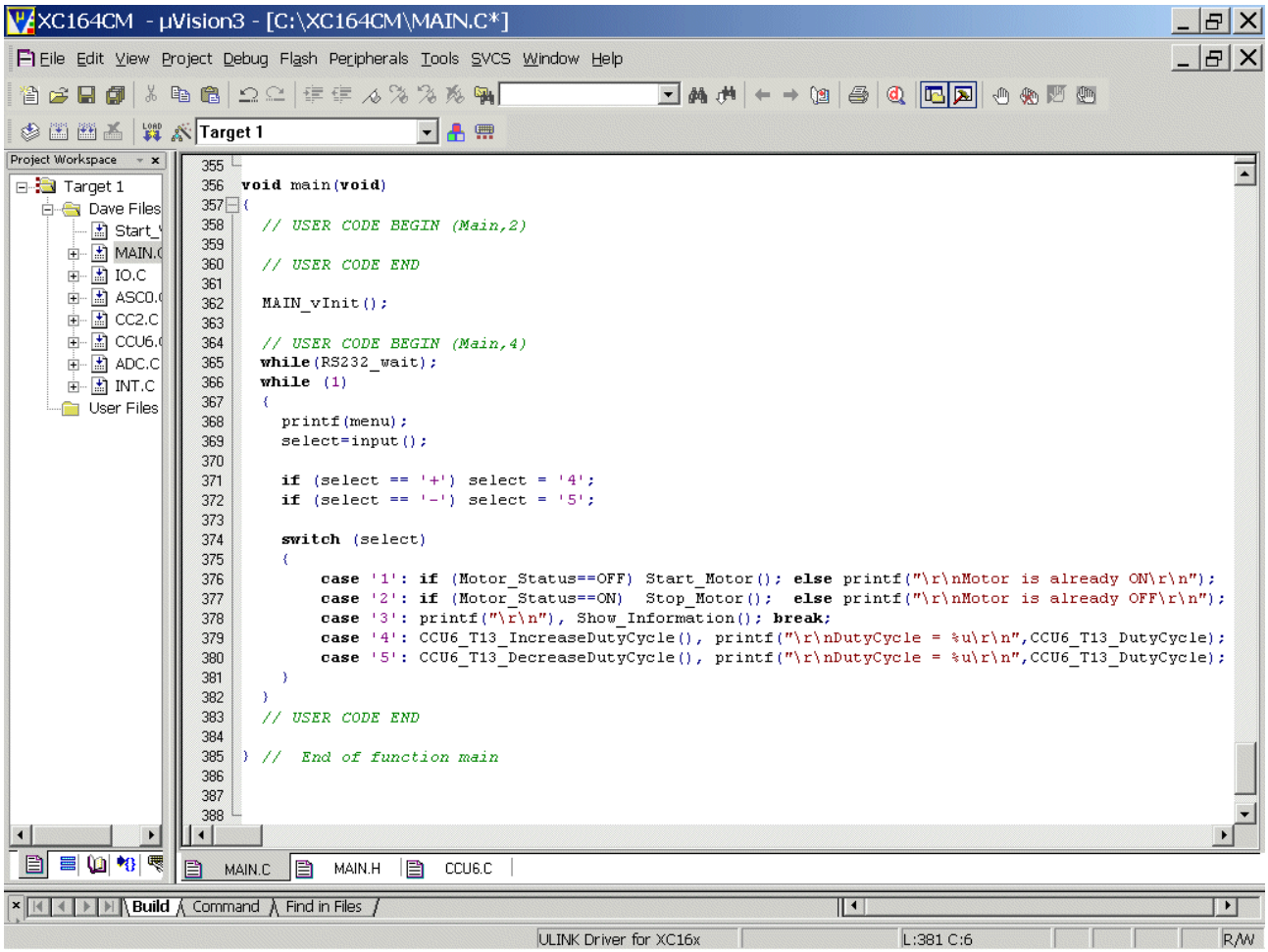

XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A6
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
334 //
335 //***
336
337 // USER CODE BEGIN (Main,1)
338 char input (void)
339 {
340 char in=' ';
341 do
342 {
343 printf(question);
344 while (!ASC0_RIC_IR);
345 ASC0_RIC_IR=0;
346 in = (char)ASC0_RBUF;
347 }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in != '-');
348 return in;
349 }
350
351 void Show_Information(void)
352 {
353 char stopit;
354
355 do
356 {
357 if (T12_Ticks_For_1_Revolution != 0)
358 printf("Speed = %5u [rpm], ",(unsigned int)(1/(T12_Ticks_For_1_Revolution*0.0000064)))
359 else
360 printf("Speed = 0 [rpm], ");
361 printf("DC Link Shunt Voltage = %5u mV \r\n",DC_Link_Current);
362 stopit=(char)toupper(ASC0_RBUF);
363 } while (stopit != 'Z');
364 }
365
366 // USER CODE END
367
MAIN.C
MAIN.H
CCU6.C
Build
Command
Find in Files
ULINK Driver for XC16x
L:350 C:1
R/W

Double click MAIN.C and change from:

```
    switch (select)
    {
        case '1': if (Motor_Status==OFF) Start_Motor(); else
printf("\r\nMotor is already ON\r\n");  break;
        case '2': if (Motor_Status==ON)  Stop_Motor();  else
printf("\r\nMotor is already OFF\r\n"); break;
        case '3': printf("\r\n"), Show_Information(); break;
    }
```

to:

```
    if (select == '+') select = '4';
    if (select == '-') select = '5';

    switch (select)
    {
        case '1': if (Motor_Status==OFF) Start_Motor(); else
printf("\r\nMotor is already ON\r\n");  break;
        case '2': if (Motor_Status==ON)  Stop_Motor();  else
printf("\r\nMotor is already OFF\r\n"); break;
        case '3': printf("\r\n"), Show_Information(); break;
        case '4': CCU6_T13_IncreaseDutyCycle(),
printf("\r\nDutyCycle = %u\r\n",CCU6_T13_DutyCycle); break;
        case '5': CCU6_T13_DecreaseDutyCycle(),
printf("\r\nDutyCycle = %u\r\n",CCU6_T13_DutyCycle); break;
    }
```

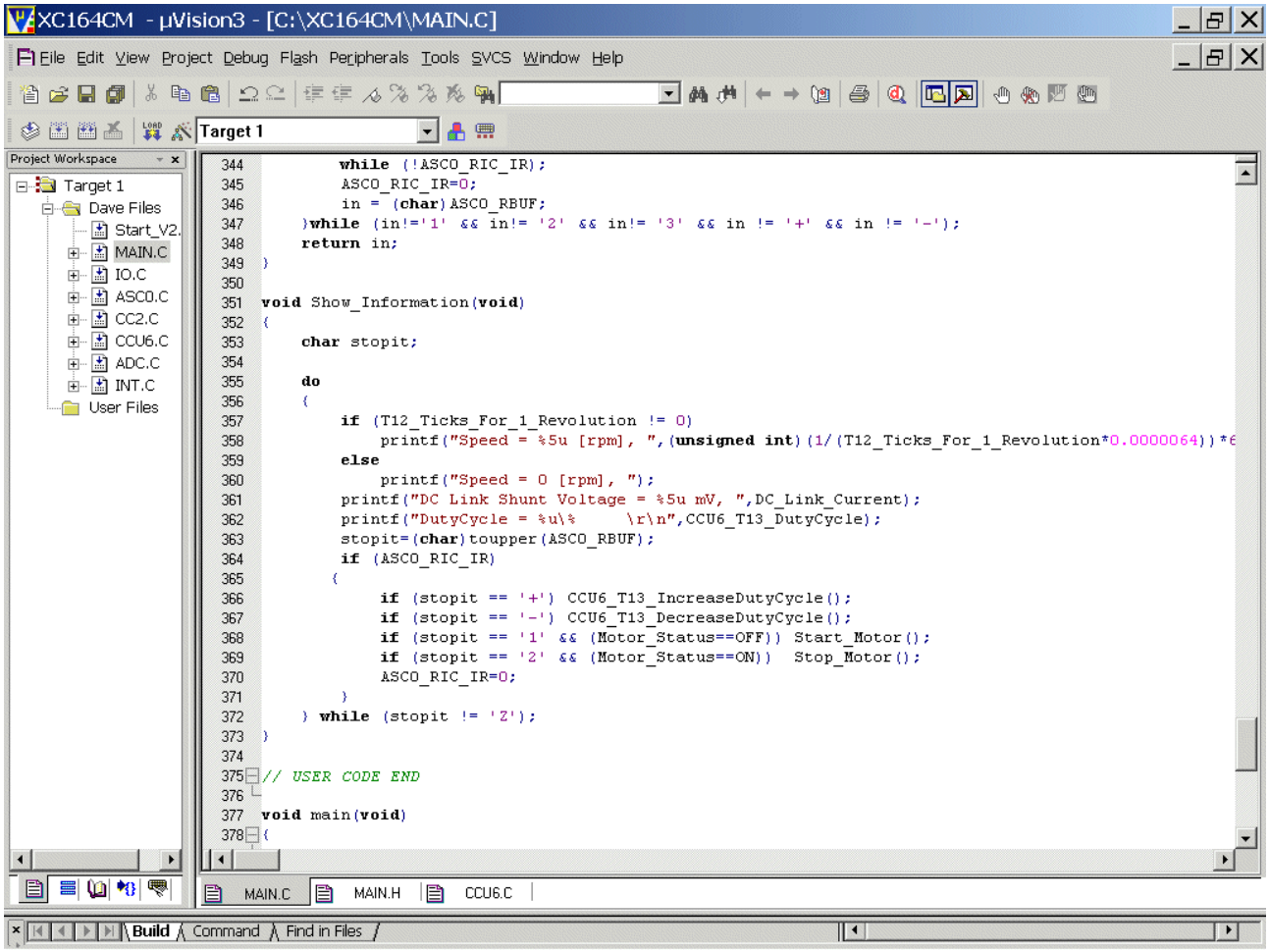

XC164CM - µVision3 - [C:\XC164CM\MAIN.C*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
355 void main(void)
356 {
357 // USER CODE BEGIN (Main,2)
358
359 // USER CODE END
360
361 MAIN_vInit();
362
363 // USER CODE BEGIN (Main,4)
364 while(RS232_wait);
365 while (1)
366 {
367 printf(menu);
368 select=input();
369
370 if (select == '+') select = '4';
371 if (select == '-') select = '5';
372
373 switch (select)
374 {
375 case '1': if (Motor_Status==OFF) Start_Motor(); else printf("\r\nMotor is already ON\r\n");
376 case '2': if (Motor_Status==ON) Stop_Motor(); else printf("\r\nMotor is already OFF\r\n");
377 case '3': printf("\r\n"), Show_Information(); break;
378 case '4': CCU6_T13_IncreaseDutyCycle(), printf("\r\nDutyCycle = %u\r\n",CCU6_T13_DutyCycle);
379 case '5': CCU6_T13_DecreaseDutyCycle(), printf("\r\nDutyCycle = %u\r\n",CCU6_T13_DutyCycle);
380 }
381 }
382 // USER CODE END
383
384 } // End of function main
385
386
387
388
MAIN.C
MAIN.H
CCU6.C
Build
Command
Find in Files
ULINK Driver for XC16x
L:381 C:6
R/W

Double click MAIN.C and change from:

```
void Show_Information(void)
{
    char stopit;

    do
    {
        if (T12_Ticks_For_1_Revolution != 0)
            printf("Speed = %5u [rpm], ",(unsigned
int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*60);
        else
            printf("Speed = 0 [rpm], ");
        printf("DC Link Shunt Voltage = %5u mV
\r\n",DC_Link_Current);
        stopit=(char)toupper(ASC0_RBUF);
    } while (stopit != 'Z');
}
```

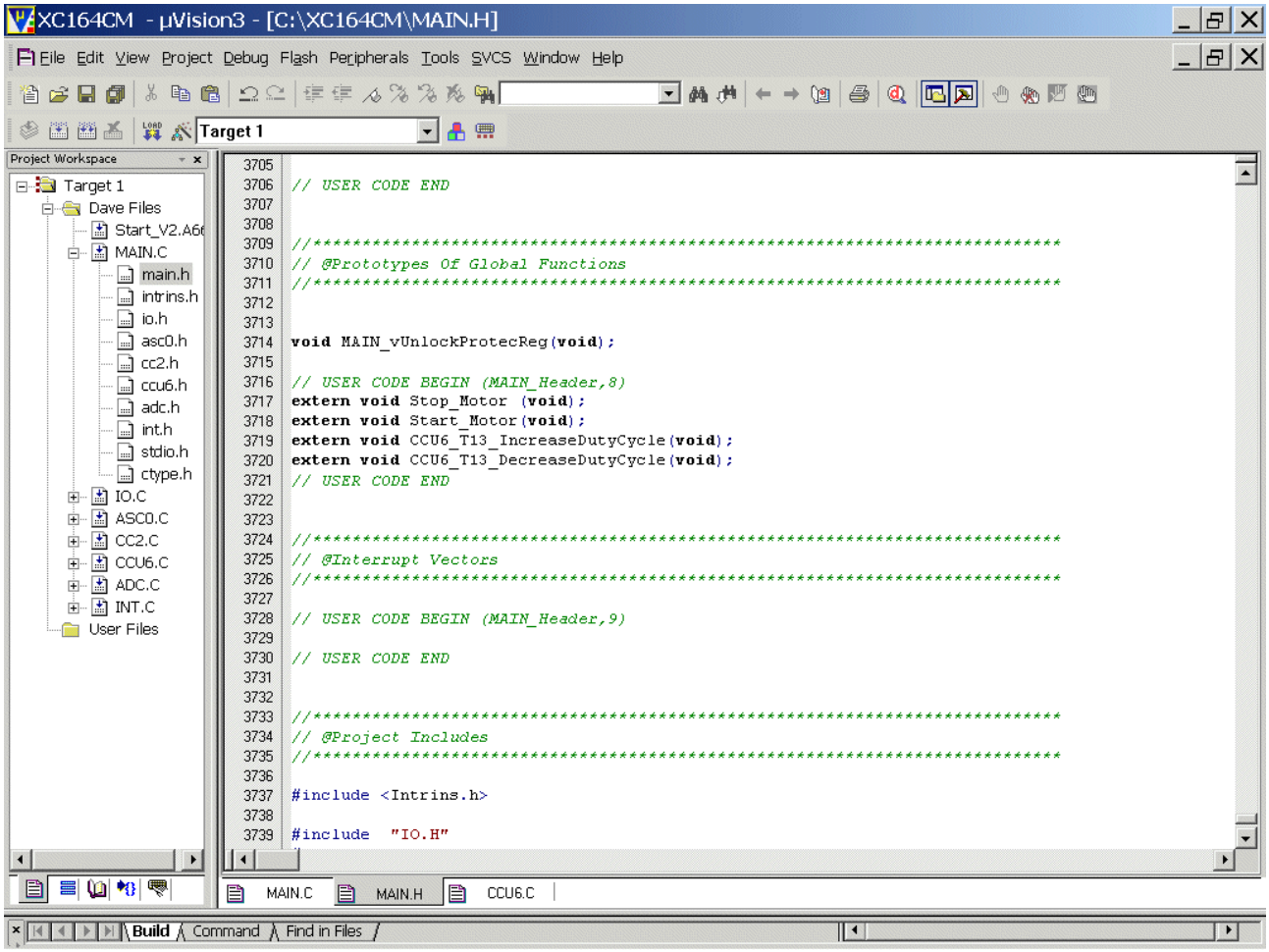
to:

```
void Show_Information(void)
{
    char stopit;

    do
    {
        if (T12_Ticks_For_1_Revolution != 0)
            printf("Speed = %5u [rpm], ",(unsigned
int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*60);
        else
            printf("Speed = 0 [rpm], ");
        printf("DC Link Shunt Voltage = %5u mV,
",DC_Link_Current);
        printf("DutyCycle = %u\%     \r\n",CCU6_T13_DutyCycle);
        stopit=(char)toupper(ASC0_RBUF);
        if (ASC0_RIC_IR)
        {
            if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
            if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
            if (stopit == '1' && (Motor_Status==OFF))
Start_Motor();
            if (stopit == '2' && (Motor_Status==ON))
Stop_Motor();
            ASC0_RIC_IR=0;
        }
    } while (stopit != 'Z');
}
```

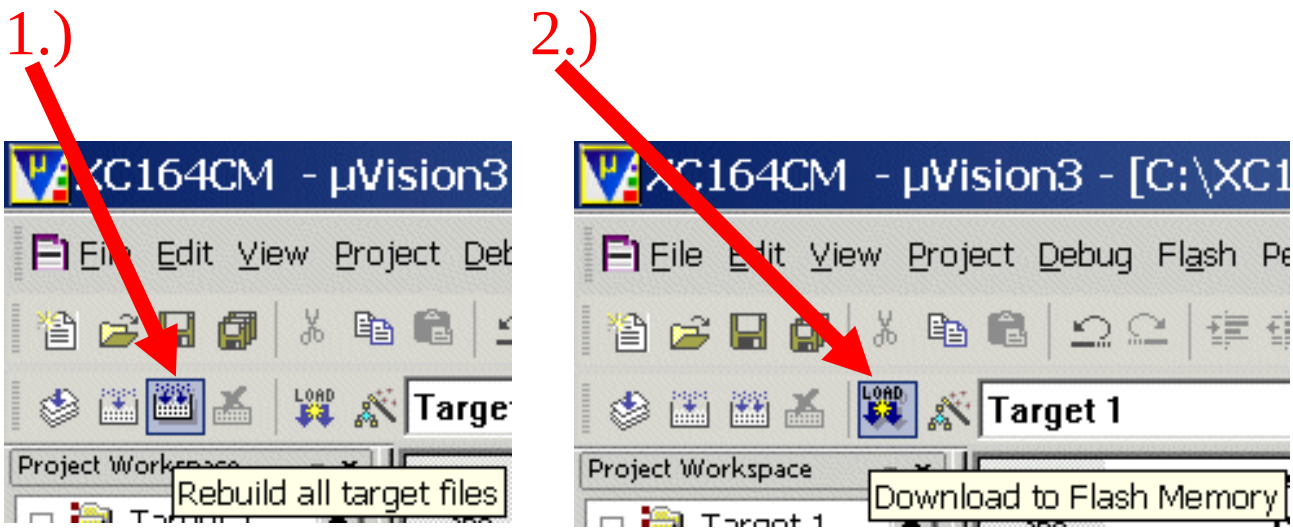

XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
344 while (!ASC0_RIC_IR);
345 ASC0_RIC_IR=0;
346 in = (char)ASC0_RBUF;
347 }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in != '-');
348 return in;
349 }
350
351 void Show_Information(void)
352 {
353 char stopit;
354
355 do
356 {
357 if (T12_Ticks_For_1_Revolution != 0)
358 printf("Speed = %5u [rpm], ",(unsigned int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*6
359 else
360 printf("Speed = 0 [rpm], ");
361 printf("DC Link Shunt Voltage = %5u mV, ",DC_Link_Current);
362 printf("DutyCycle = %u\% \r\n",CCU6_T13_DutyCycle);
363 stopit=(char)toupper(ASC0_RBUF);
364 if (ASC0_RIC_IR)
365 {
366 if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
367 if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
368 if (stopit == '1' && (Motor_Status==OFF)) Start_Motor();
369 if (stopit == '2' && (Motor_Status==ON)) Stop_Motor();
370 ASC0_RIC_IR=0;
371 }
372 } while (stopit != 'Z');
373 }
374
375 // USER CODE END
376
377 void main(void)
378 {
MAIN.C
MAIN.H
CCU6.C
Build
Command
Find in Files

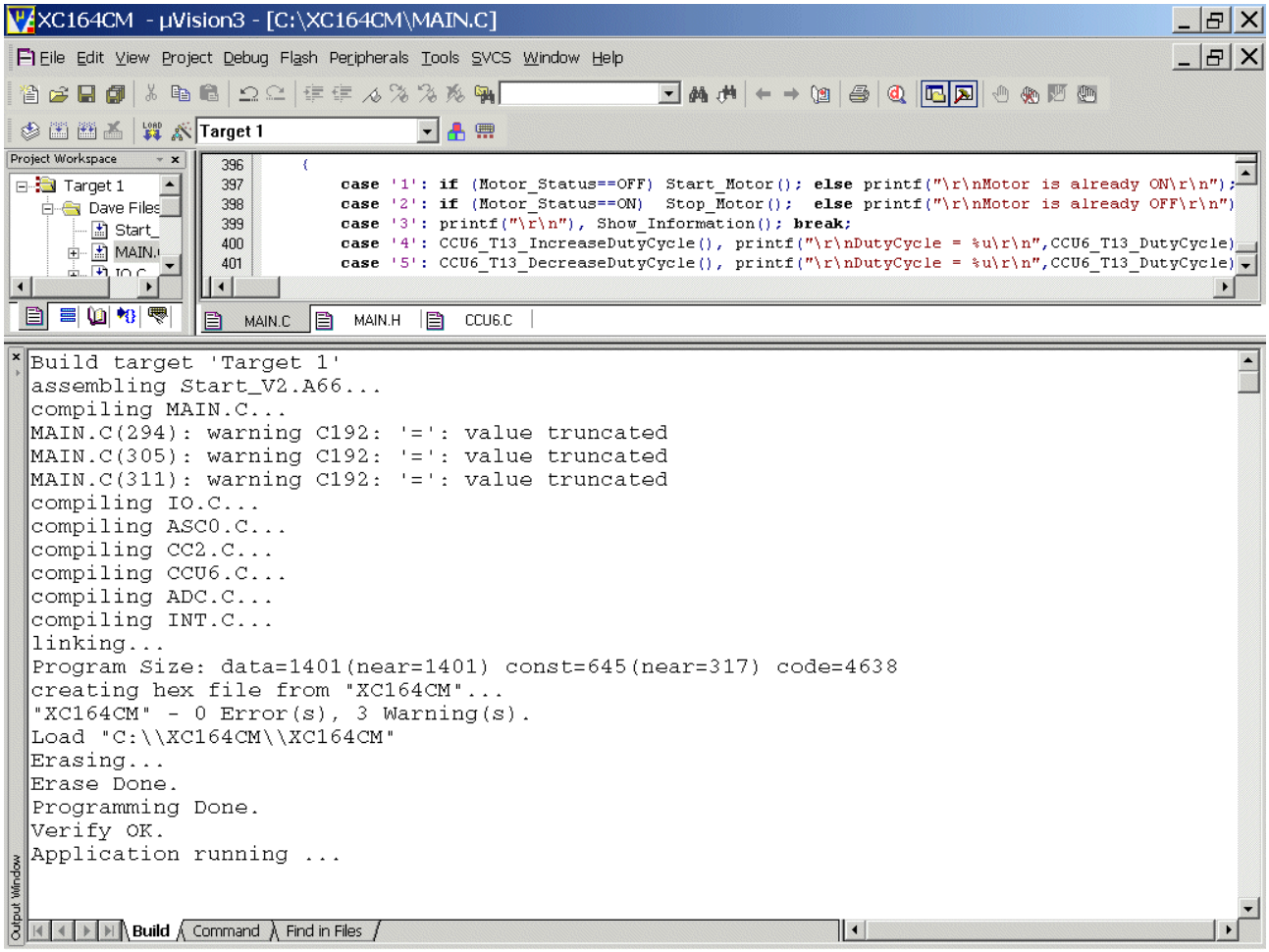
Double click main.h and DELETE Extern Declaration of Prototype of Global Function:

```
extern void Commutation(void);
```



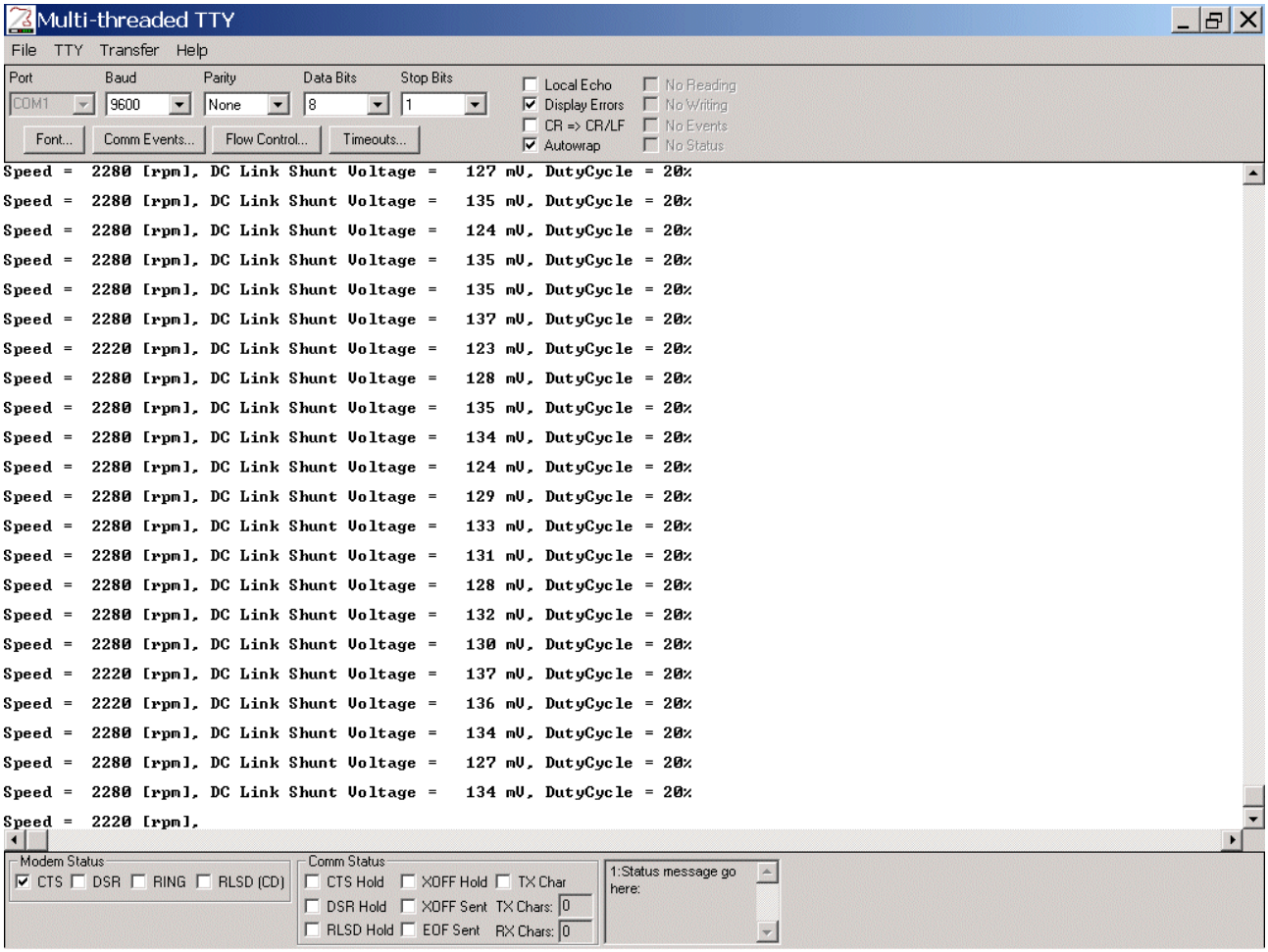
Build Application:





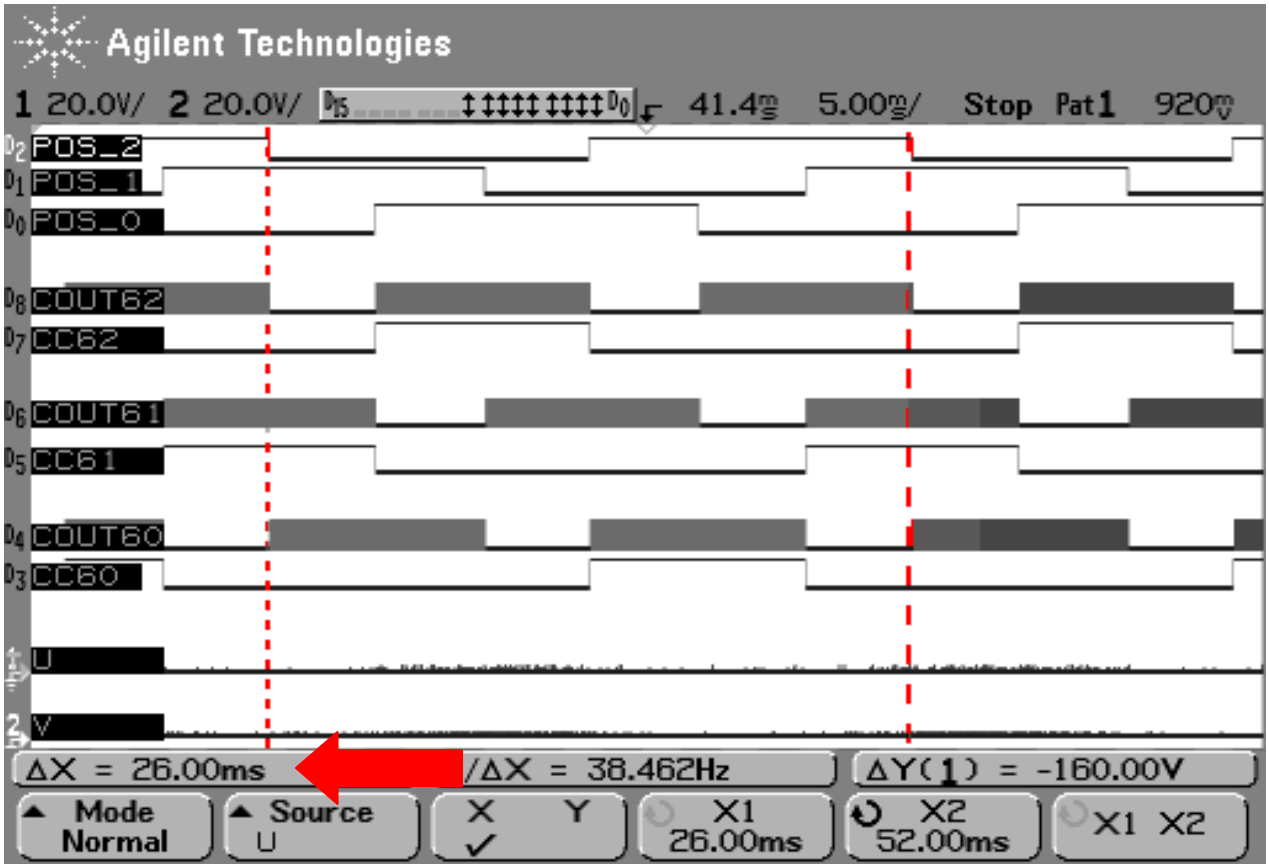
Run application:

e.g. DutyCycle = 20 %

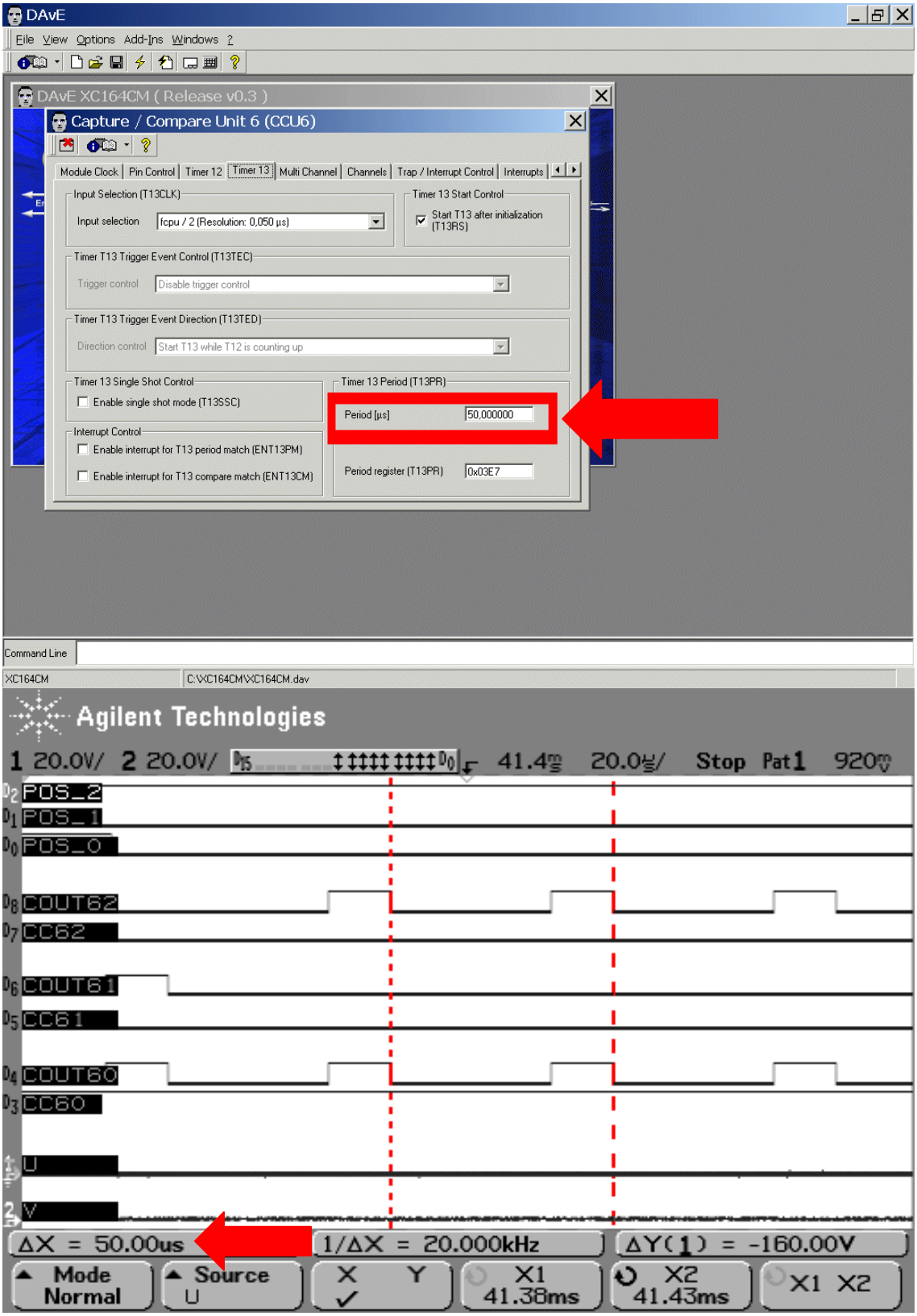


e.g. DutyCycle = 20 %

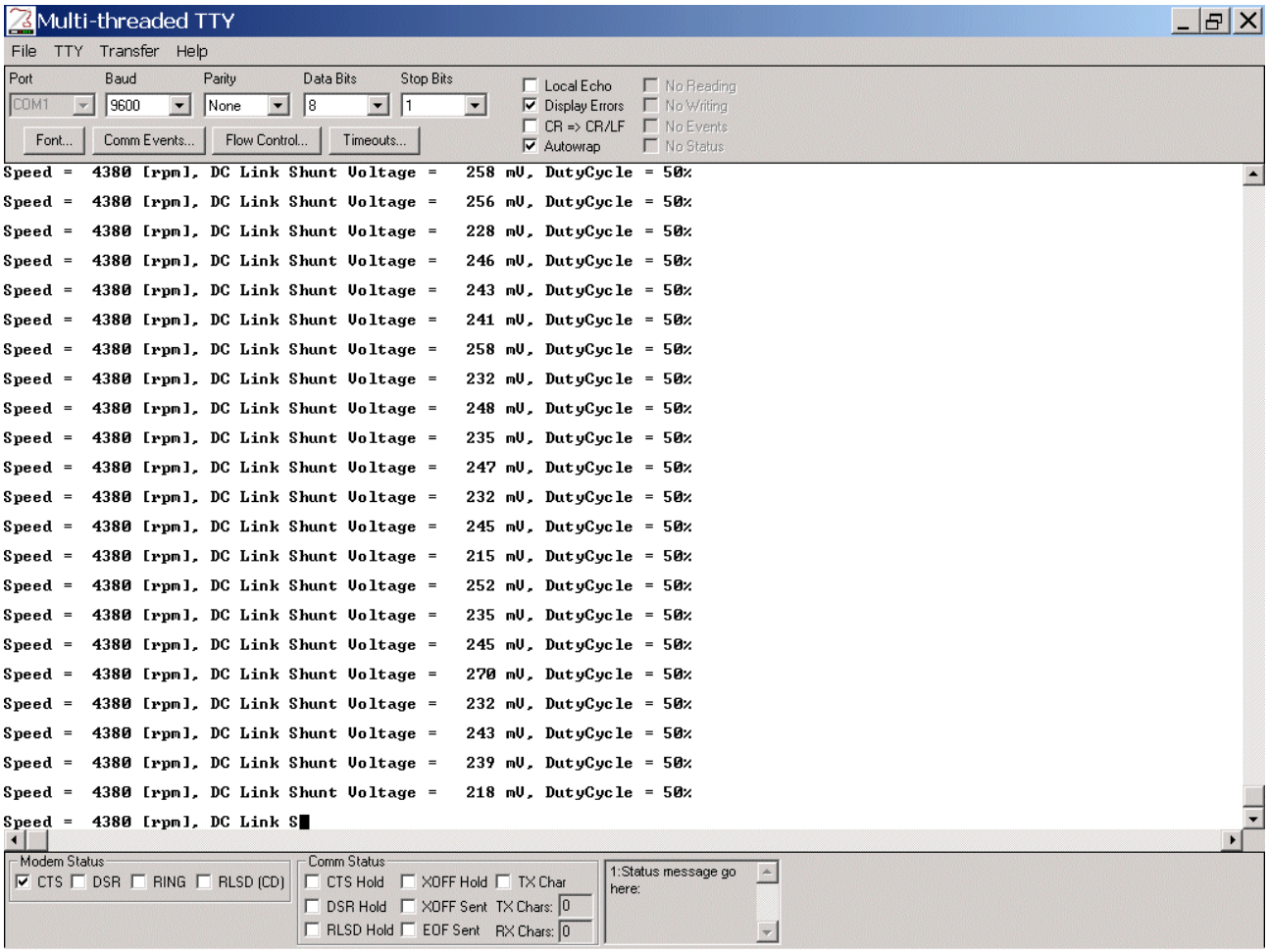
revolutions per minute [rpm]	revolutions per second [rps]	time for 1 revolution [ms]	time for 1 commutation window [ms]	commutation windows per second []
1000	16,67	60	10,00	100
2000	33,33	30	5,00	200
2280	38,00	26	4,39	228
3000	50,00	20	3,33	300
4000	66,67	15	2,50	400
5000	83,33	12	2,00	500
6000	100,00	10	1,67	600



e.g. DutyCycle = 20 %

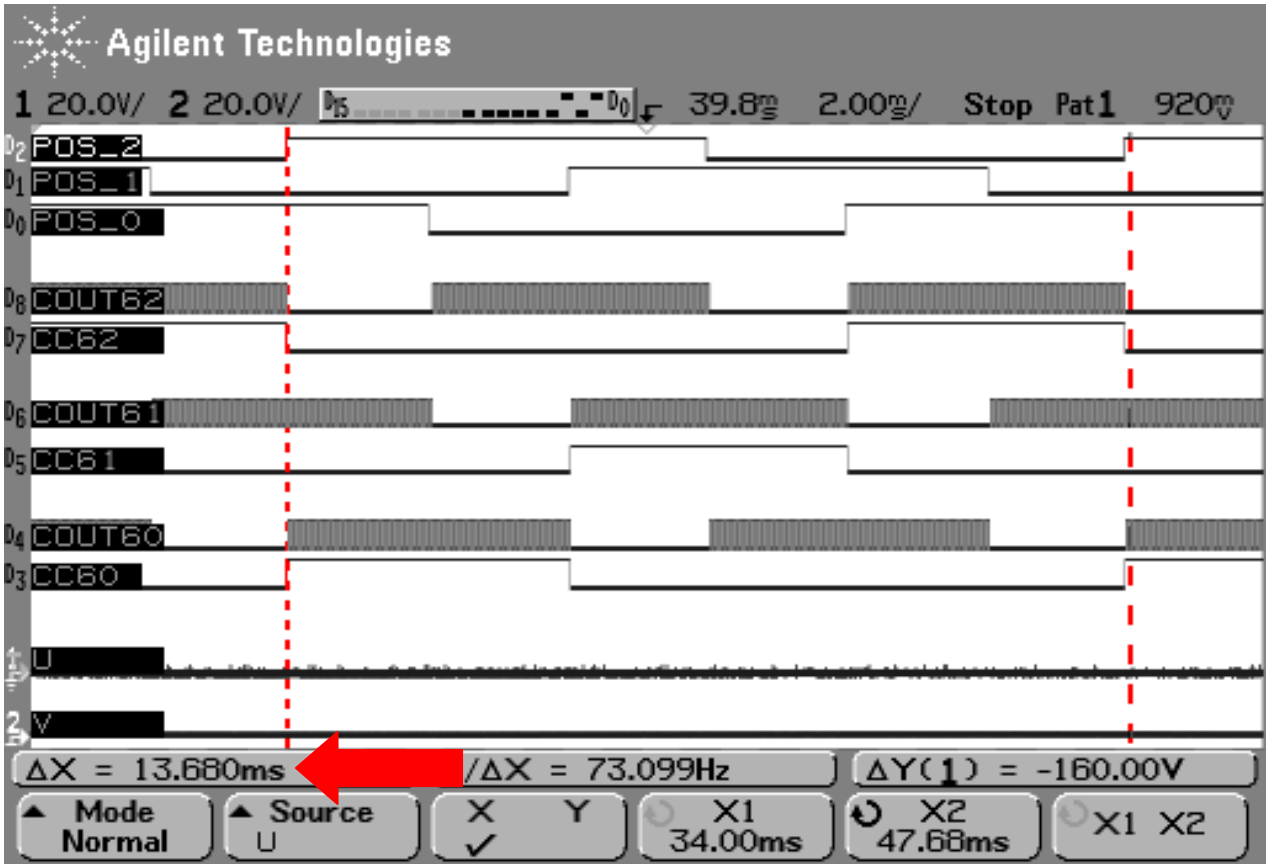


e.g. DutyCycle = 50 %

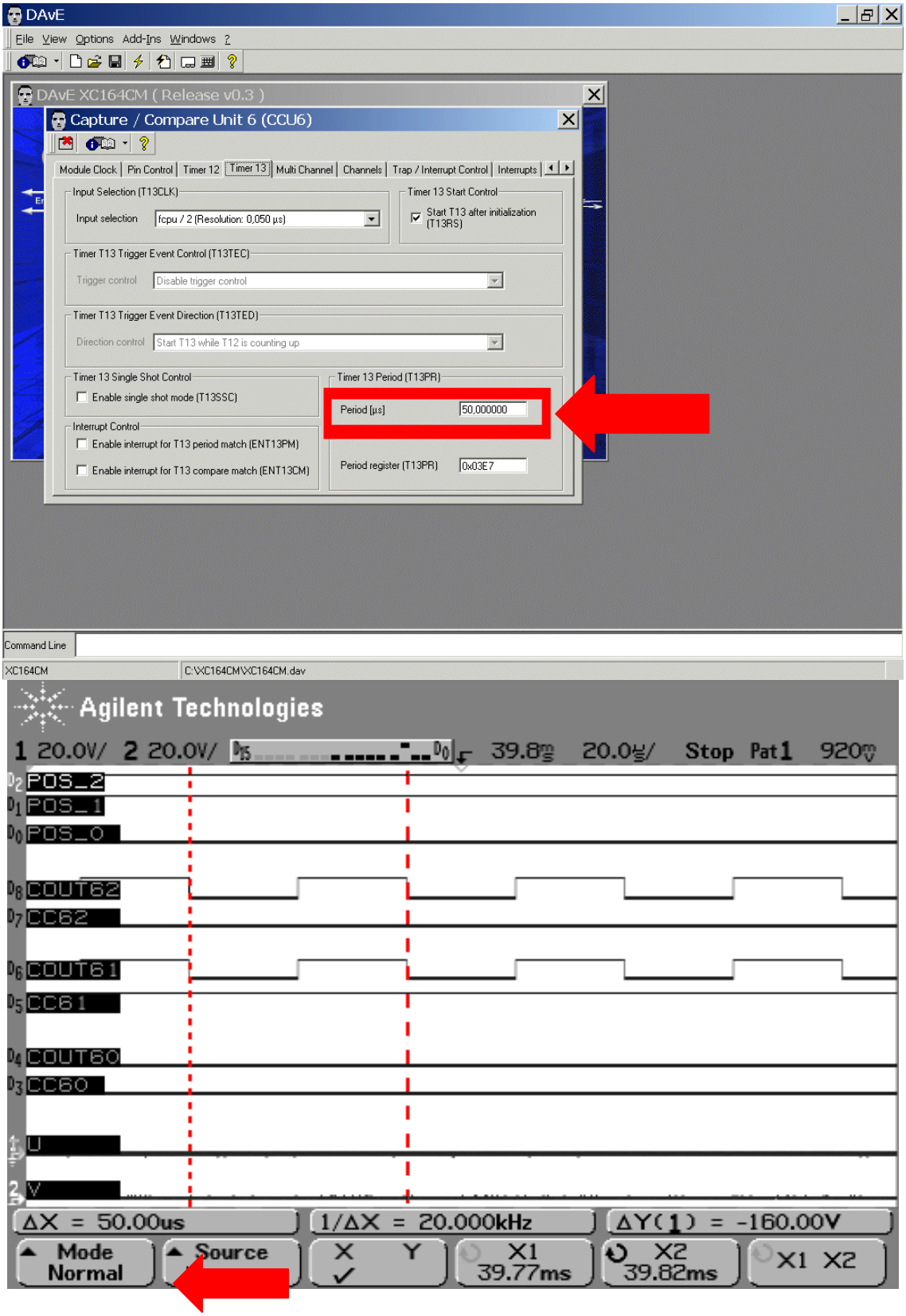


e.g. DutyCycle = 50 %

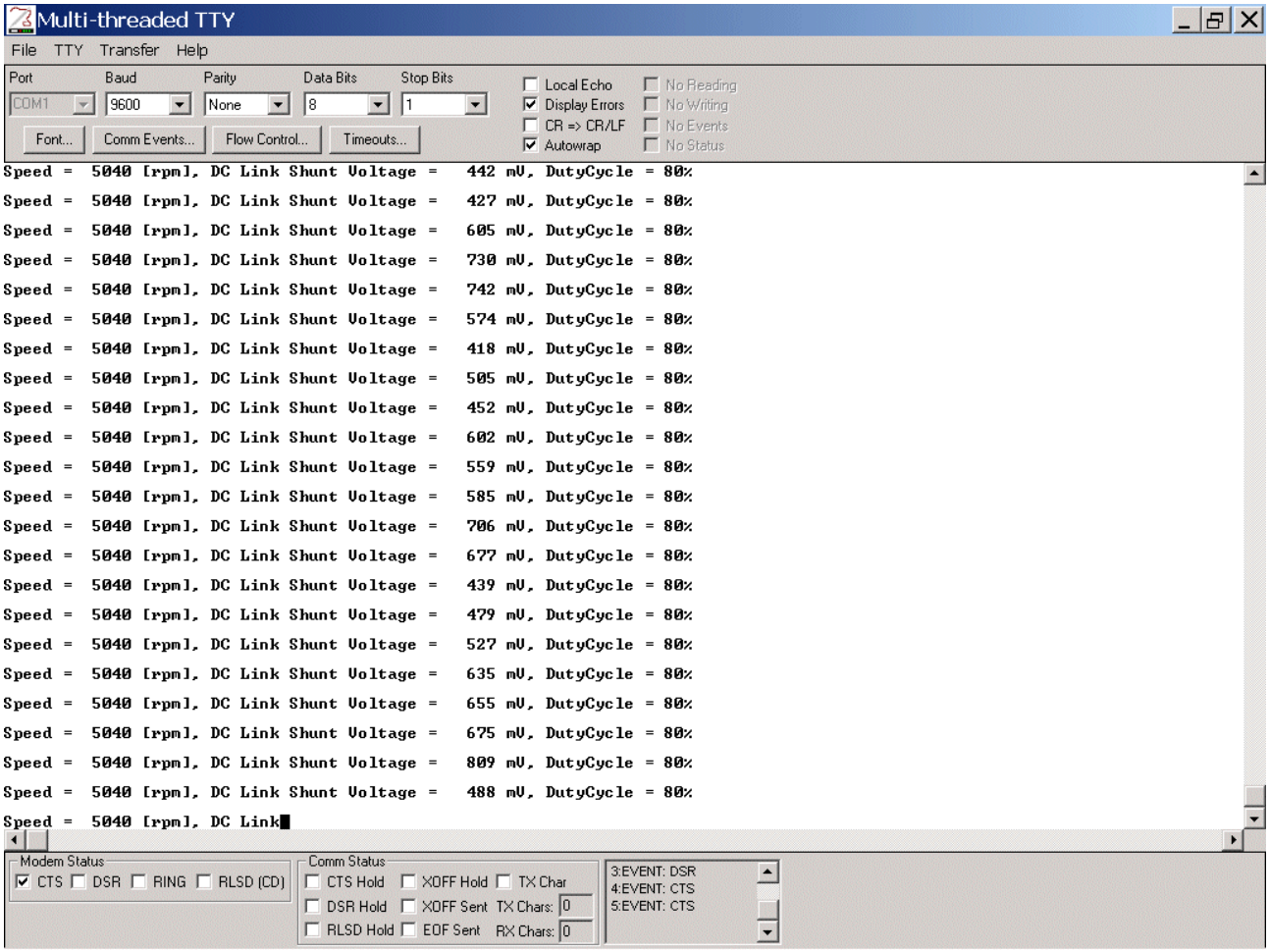
revolutions per minute [rpm]	revolutions per second [rps]	time for 1 revolution [ms]	time for 1 commutation window [ms]	commutation windows per second []
1000	16,67	60	10,00	100
2000	33,33	30	5,00	200
3000	50,00	20	3,33	300
4000	66,67	15	2,50	400
4380	73,00	13,699	2,28	438
5000	83,33	12	2,00	500
6000	100,00	10	1,67	600



e.g. DutyCycle = 50 %

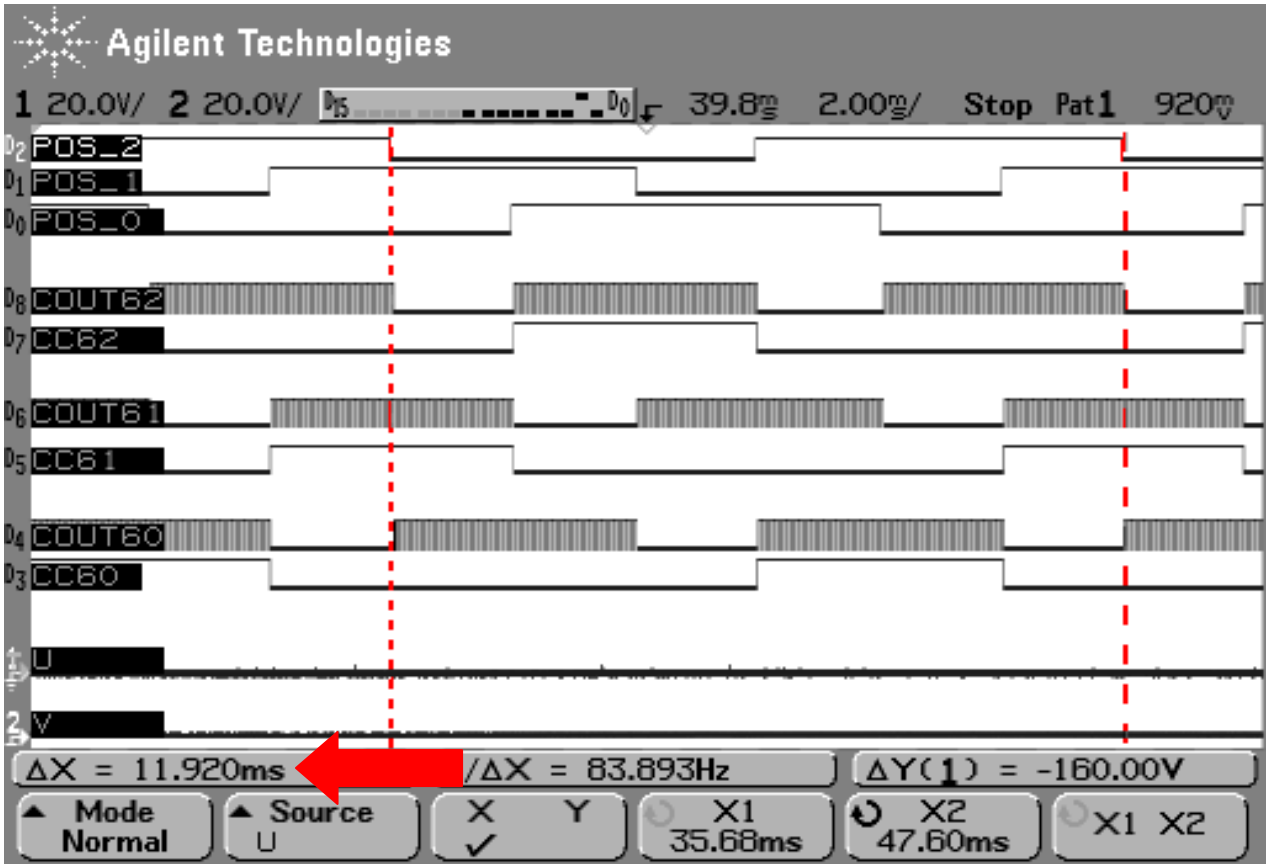


e.g. DutyCycle = 80 %

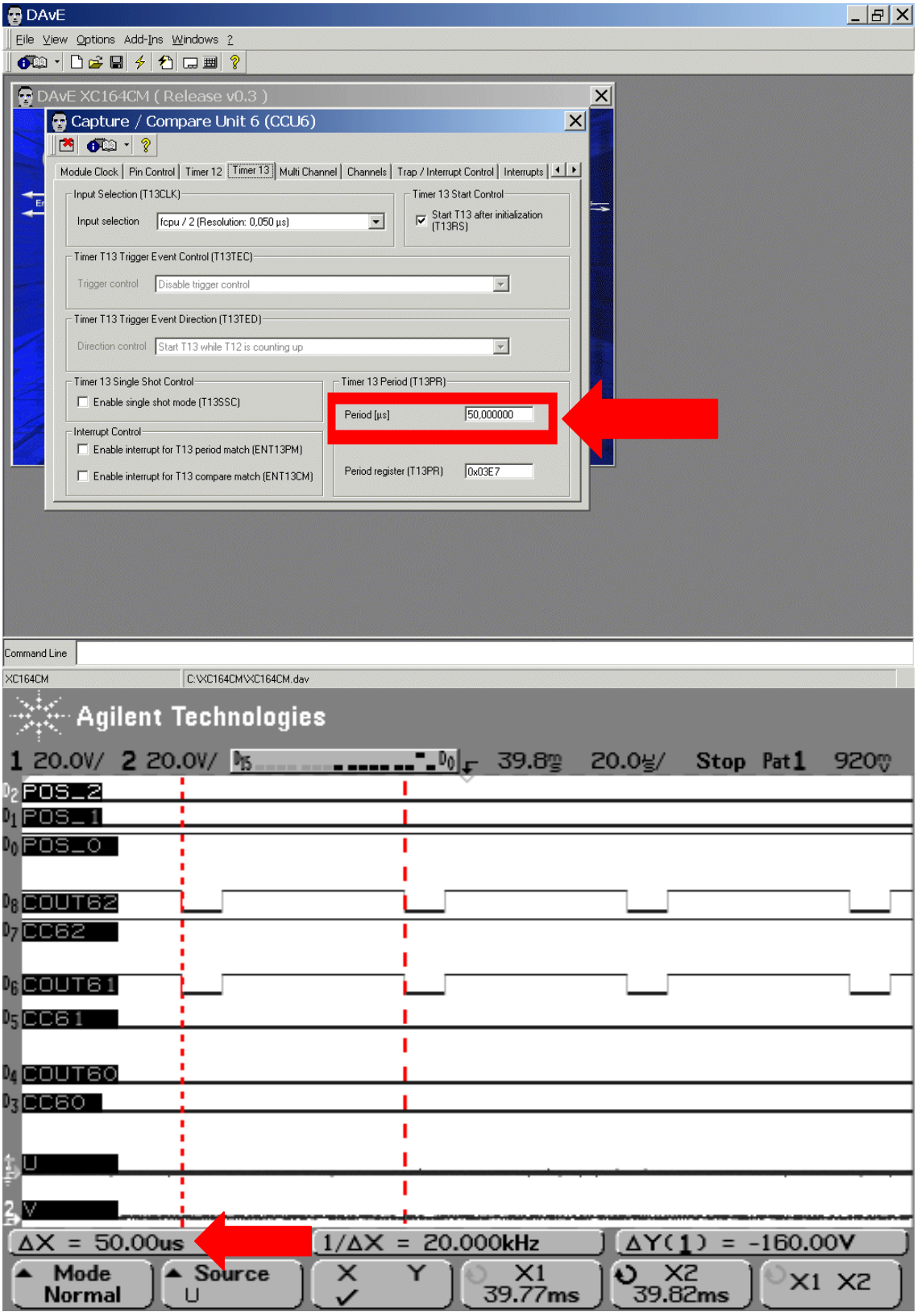


e.g. DutyCycle = 80 %

revolutions per minute [rpm]	revolutions per second [rps]	time for 1 revolution [ms]	time for 1 commutation window [ms]	commutation windows per second []
1000	16,67	60	10,00	100
2000	33,33	30	5,00	200
3000	50,00	20	3,33	300
4000	66,67	15	2,50	400
5000	83,33	12	2,00	500
5040	84,00	11,90	1,98	504
6000	100,00	10	1,67	600



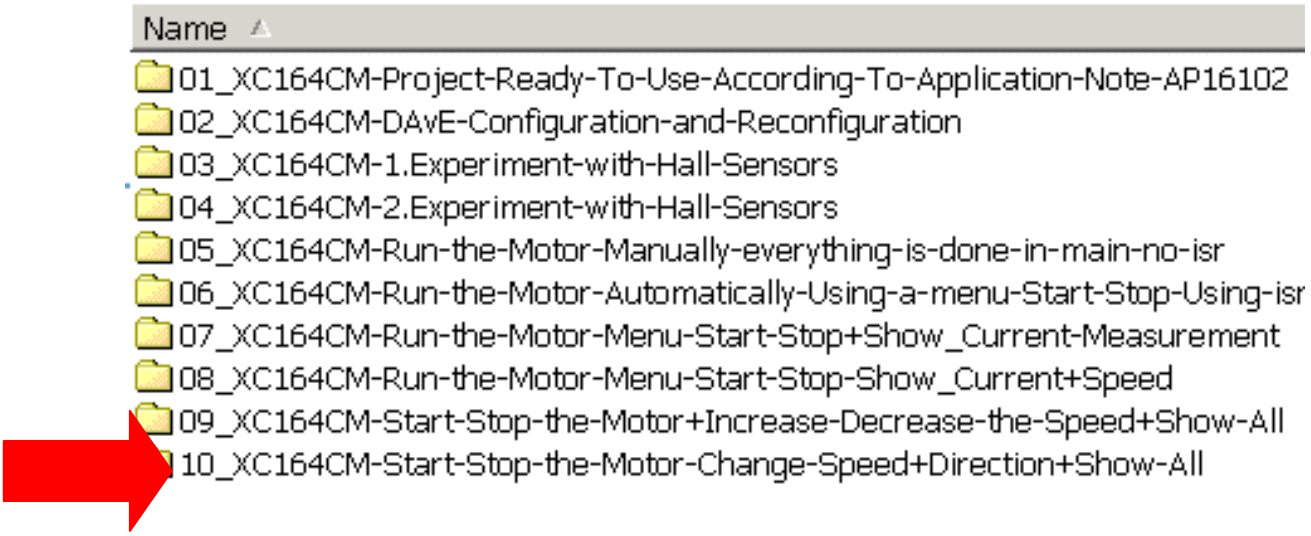
e.g. DutyCycle = 80 %

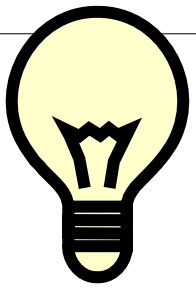


10.)

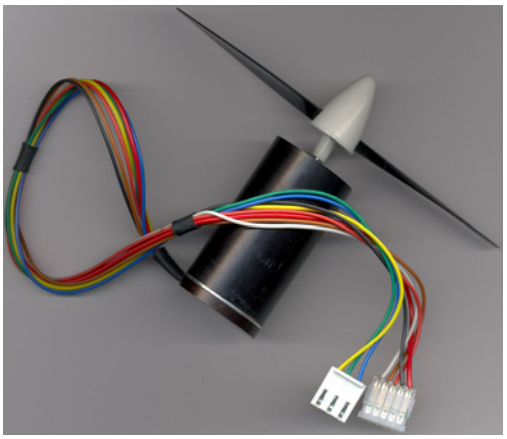
Changing
The Running Direction Of The Motor:

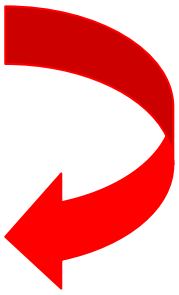
Relevant Project:





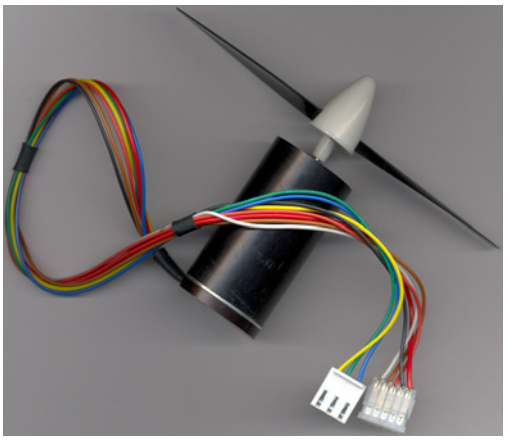
Register Information / Additional Information:





```
// ROTATE RIGHT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateRight[]   = { 000,  015,  023,  031,  046,  054,  062 };
```

```
// ROTATE RIGHT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateRight[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```

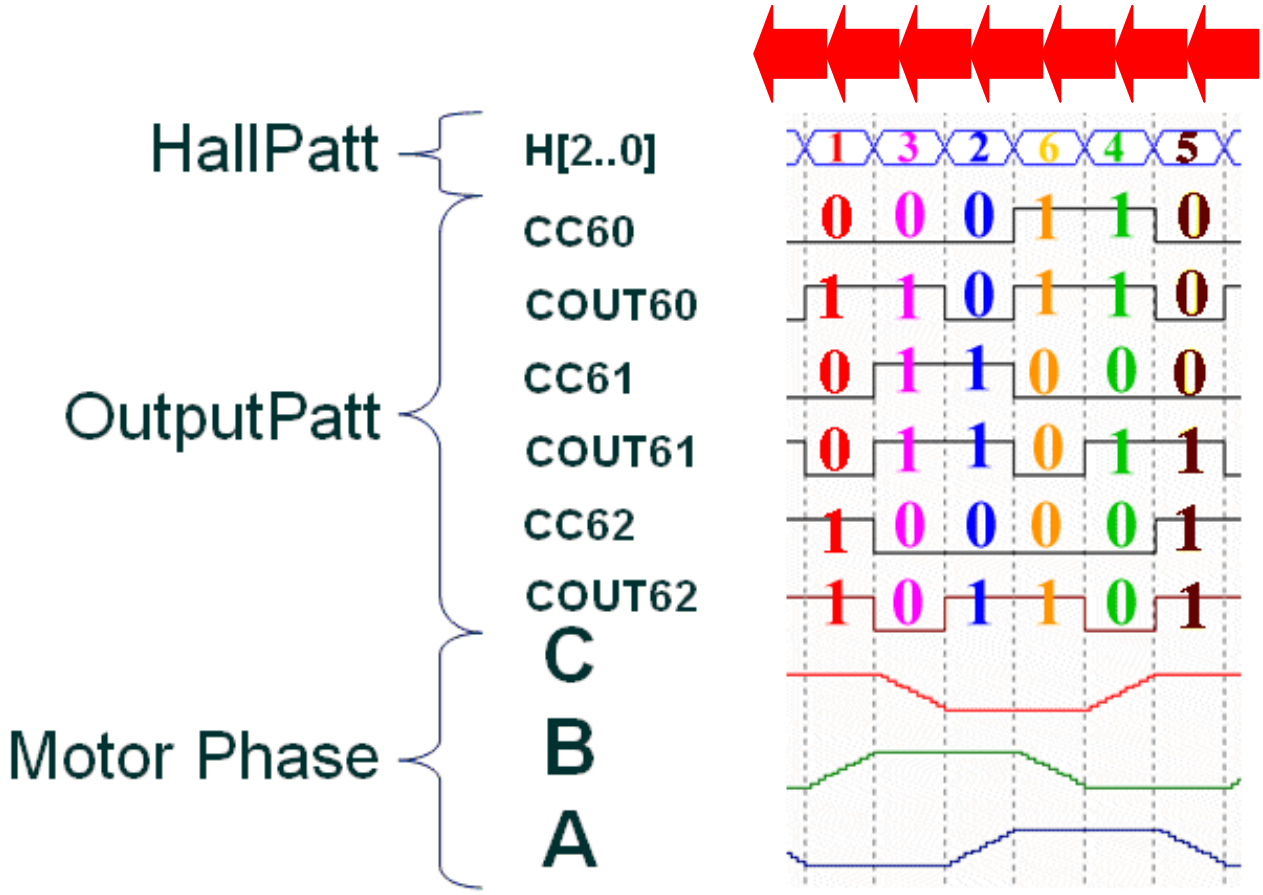


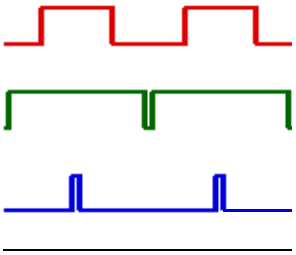
```
// ROTATE LEFT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateLeft[]   = { 000,  013,  026,  032,  045,  051,  064 };
```

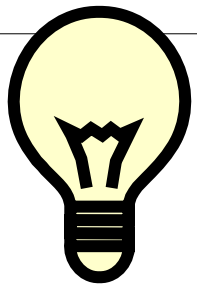
```
// ROTATE LEFT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateLeft[] = {0x00, 0x23, 0x38, 0x0B, 0x0E, 0x2C, 0x32,};
```

```
// ROTATE RIGHT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateRight[]   = { 000,  015,  023,  031,  046,  054,  062 };
```

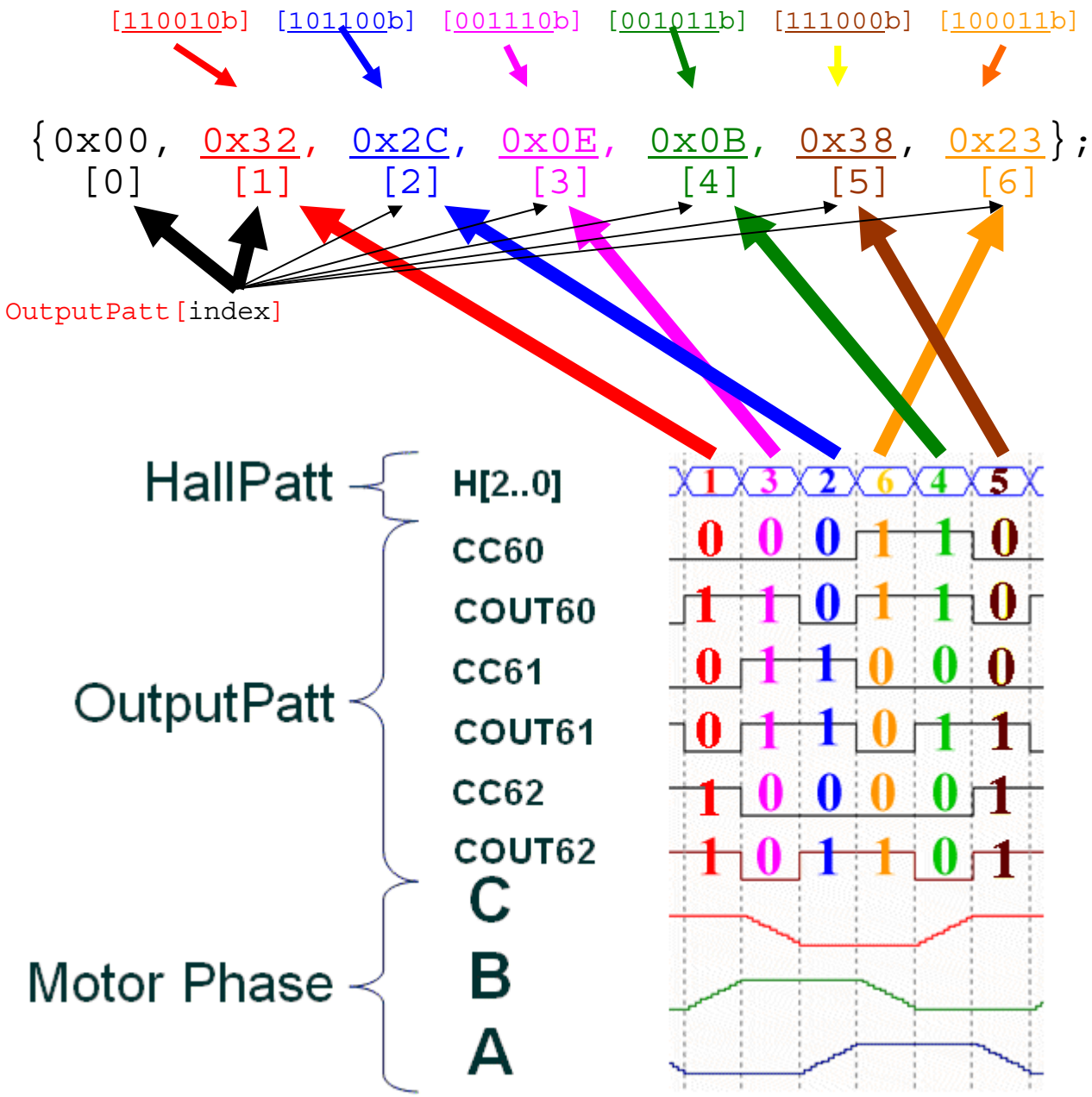
00⇨0, 01⇨5, 02⇨3, 03⇨1, 04⇨6, 05⇨4, 06⇨2





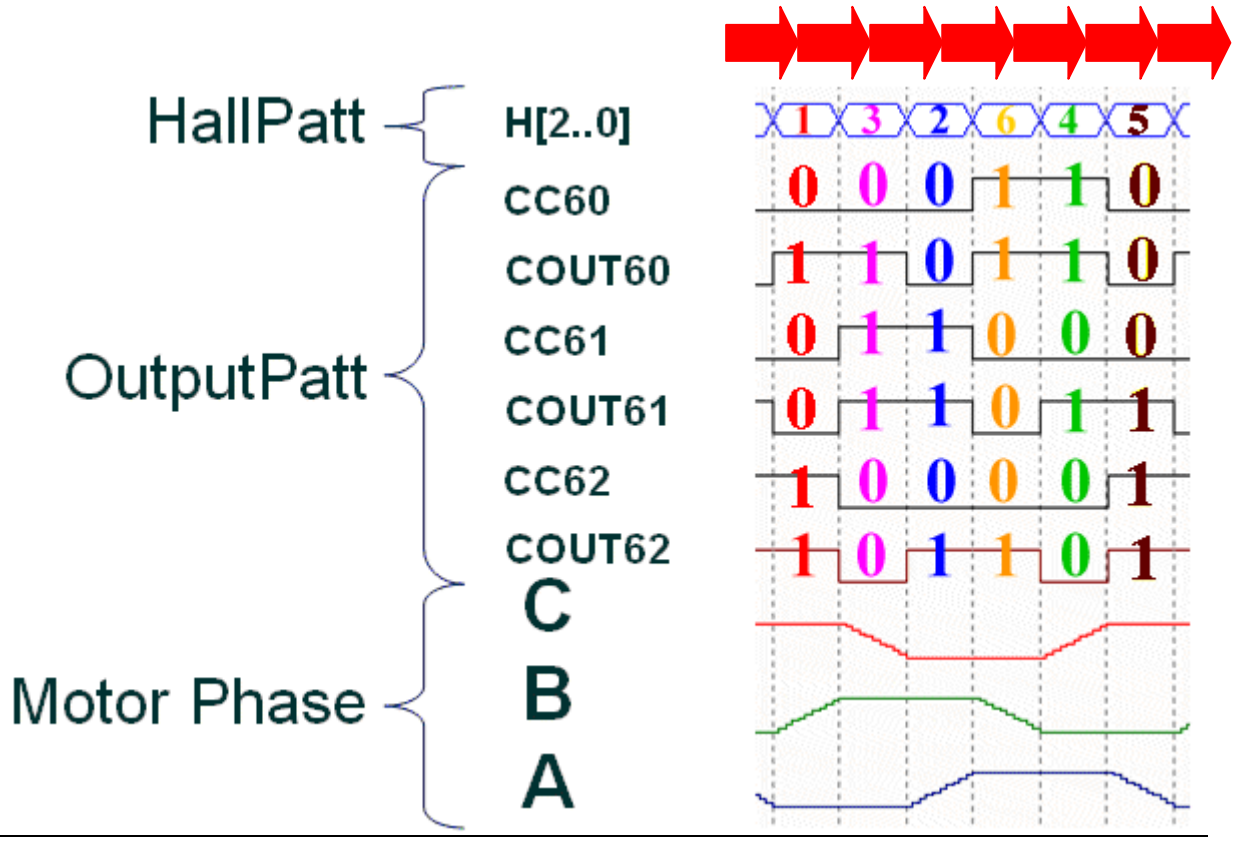


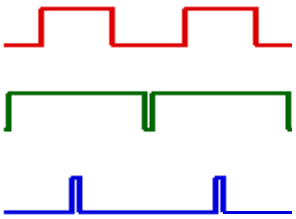
```
// ROTATE RIGHT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateRight[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```

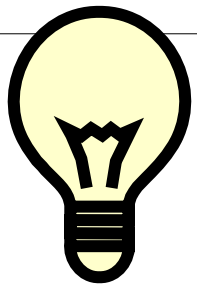


```
// ROTATE LEFT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateLeft[]   = { 000,  013,  026,  032,  045,  051,  064 };
```

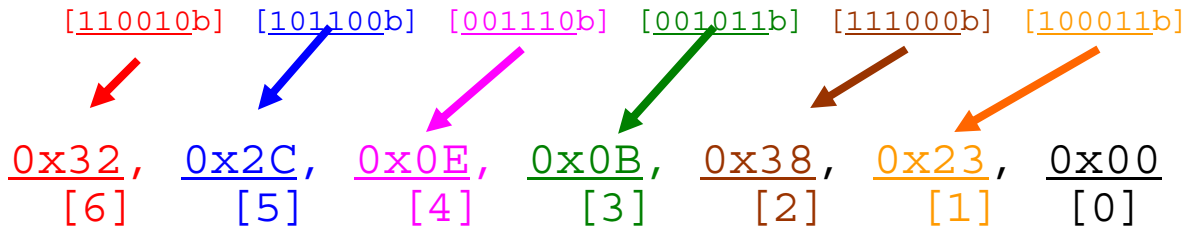
00⇨0, 01⇨3, 02⇨6, 03⇨2, 04⇨5, 05⇨1, 06⇨4

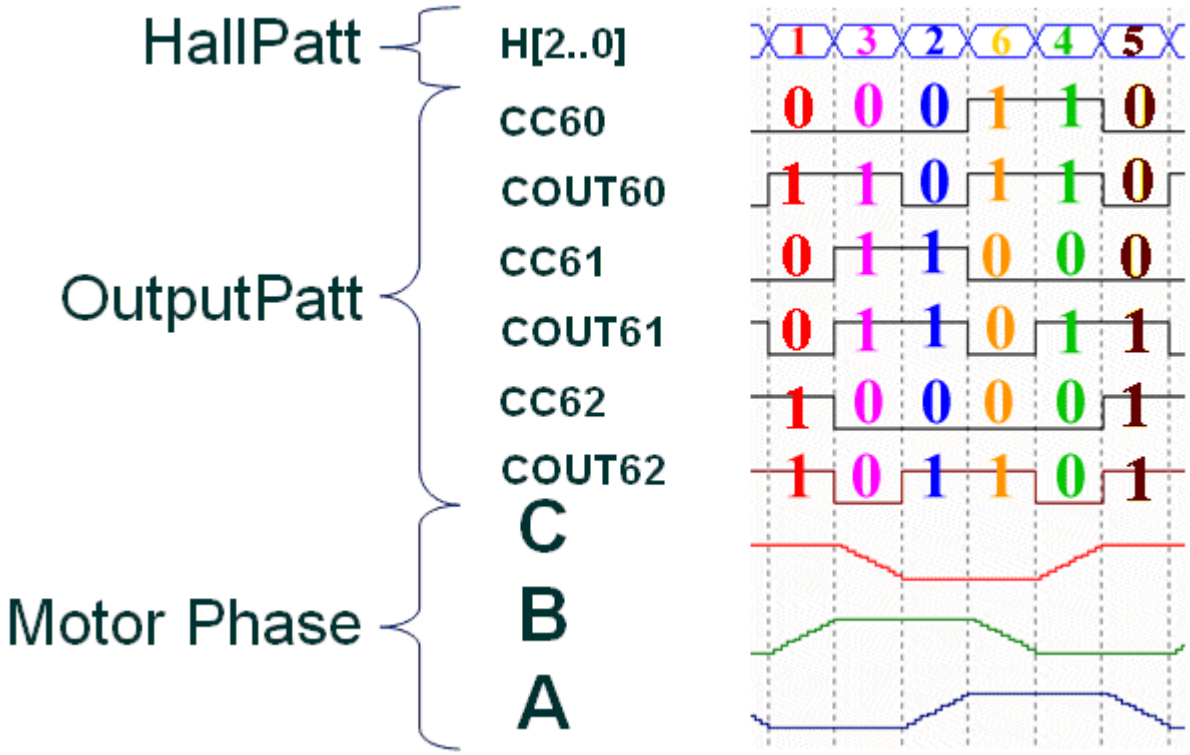






```
// ROTATE LEFT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateLeft[] = {0x00, 0x23, 0x38, 0x0B, 0x0E, 0x2C, 0x32,};
```



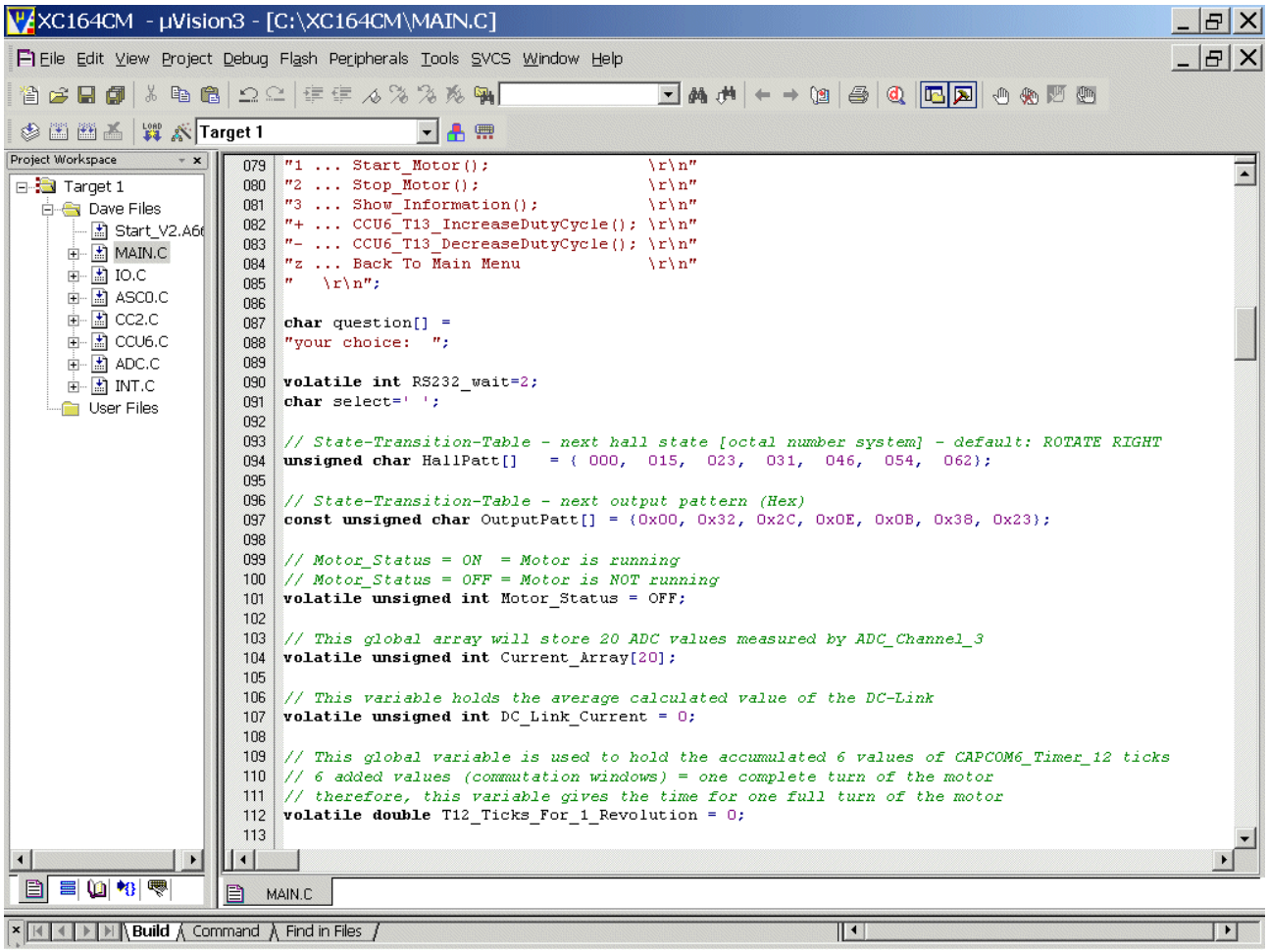


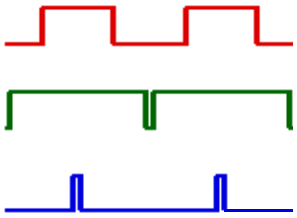
Double click MAIN.C and change Definition of Global Variable from:

```
// State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt[]   = { 000,  015,  023,  031,  046,  054,  062 };
```

to:

```
// State-Transition-Table - next hall state [octal number system] – default: ROTATE RIGHT
unsigned char HallPatt[]   = { 000,  015,  023,  031,  046,  054,  062 };
```



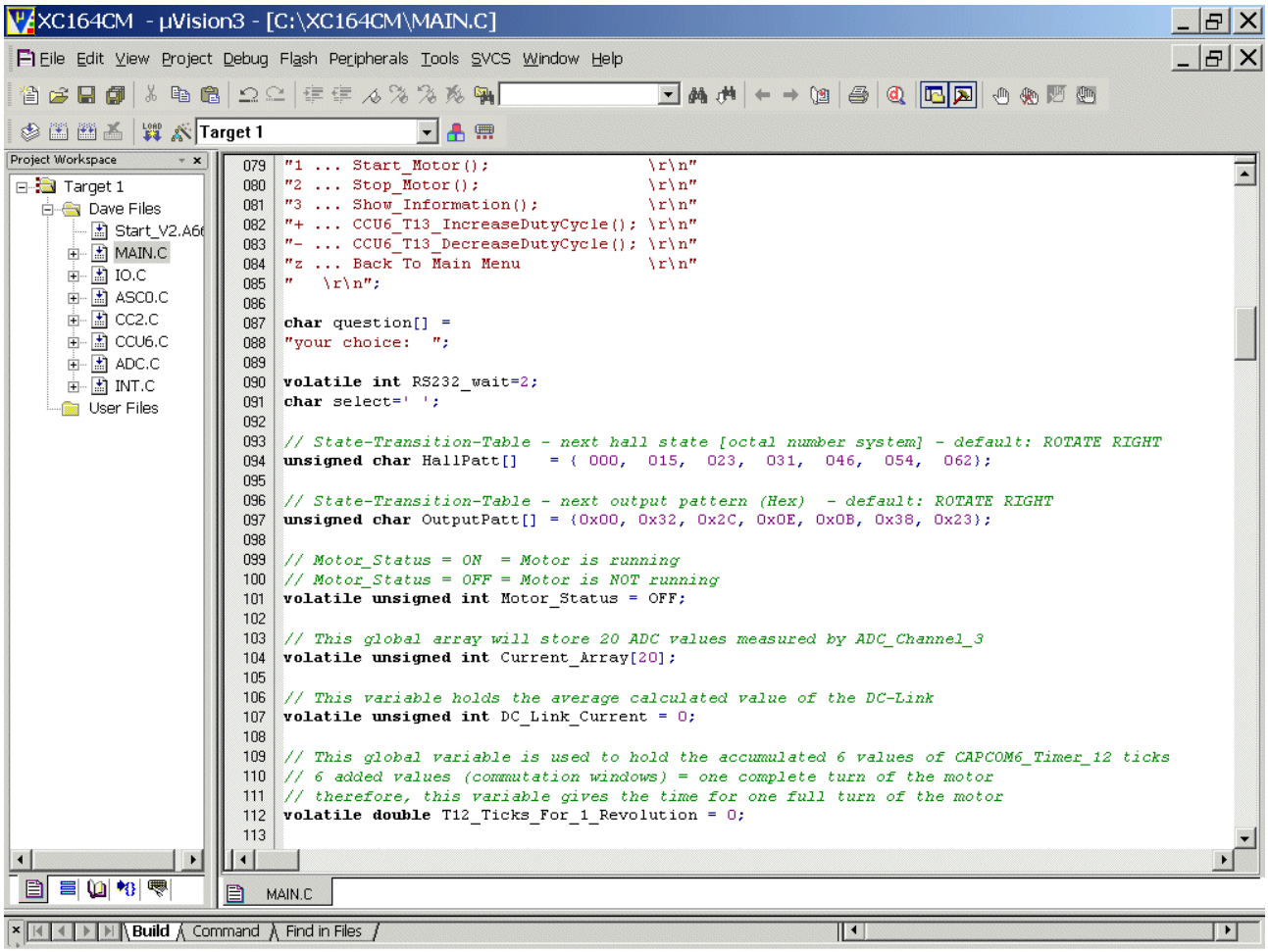


Double click MAIN.C and change Definition of Global Variable from:

```
// State-Transition-Table - next output pattern (Hex)
const unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```

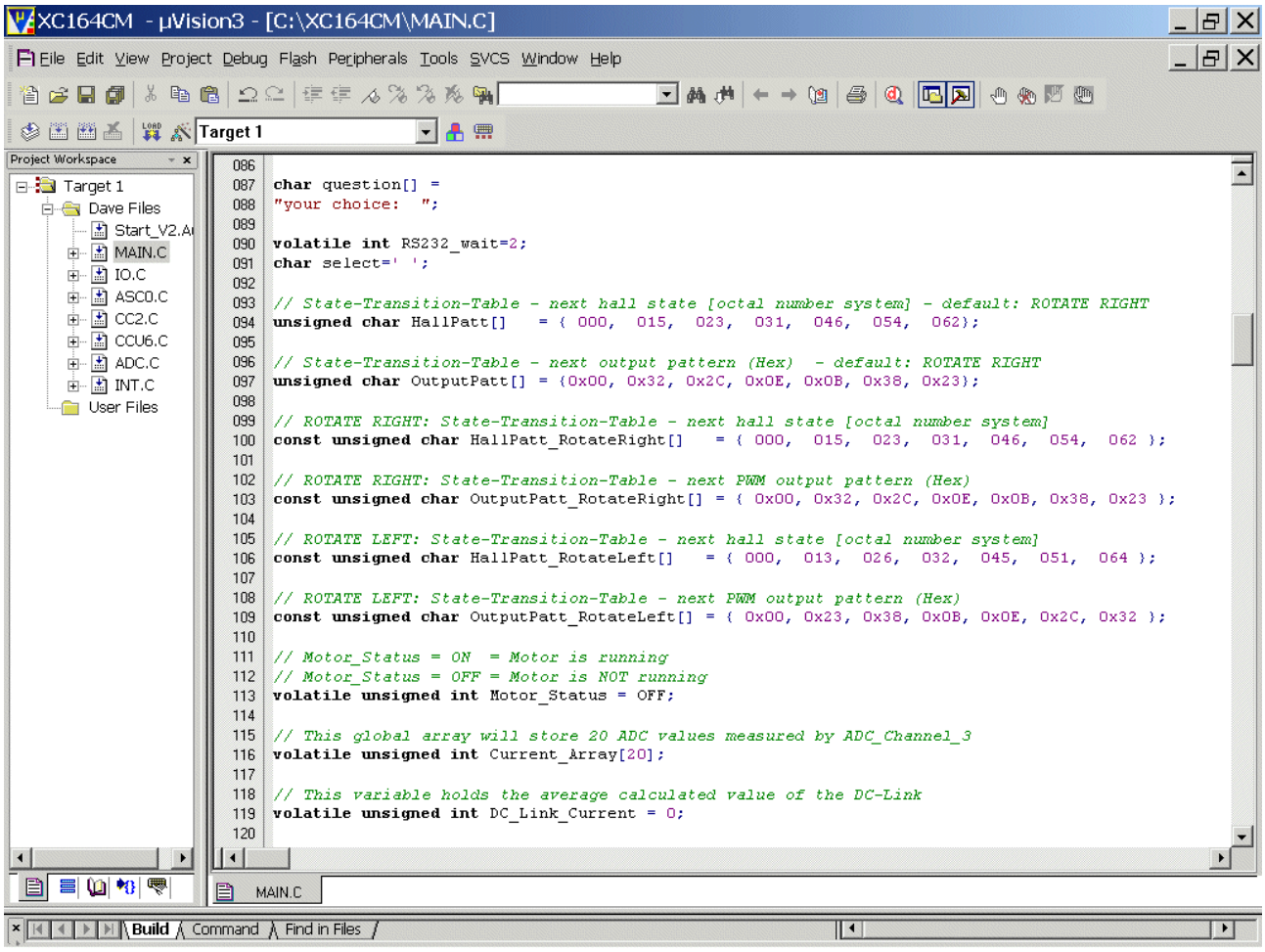
to:

```
// State-Transition-Table - next output pattern (Hex)  – default: ROTATE RIGHT
unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
```



Double click MAIN.C and insert Definition of Global Variables:

```
// ROTATE RIGHT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateRight[]   = { 000,  015,  023,  031,  046,  054,  062 };

// ROTATE RIGHT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateRight[] = { 0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23 };

// ROTATE LEFT: State-Transition-Table - next hall state [octal number system]
const unsigned char HallPatt_RotateLeft[]   = { 000,  013,  026,  032,  045,  051,  064 };

// ROTATE LEFT: State-Transition-Table - next PWM output pattern (Hex)
const unsigned char OutputPatt_RotateLeft[] = { 0x00, 0x23, 0x38, 0x0B, 0x0E, 0x2C, 0x32 };
```

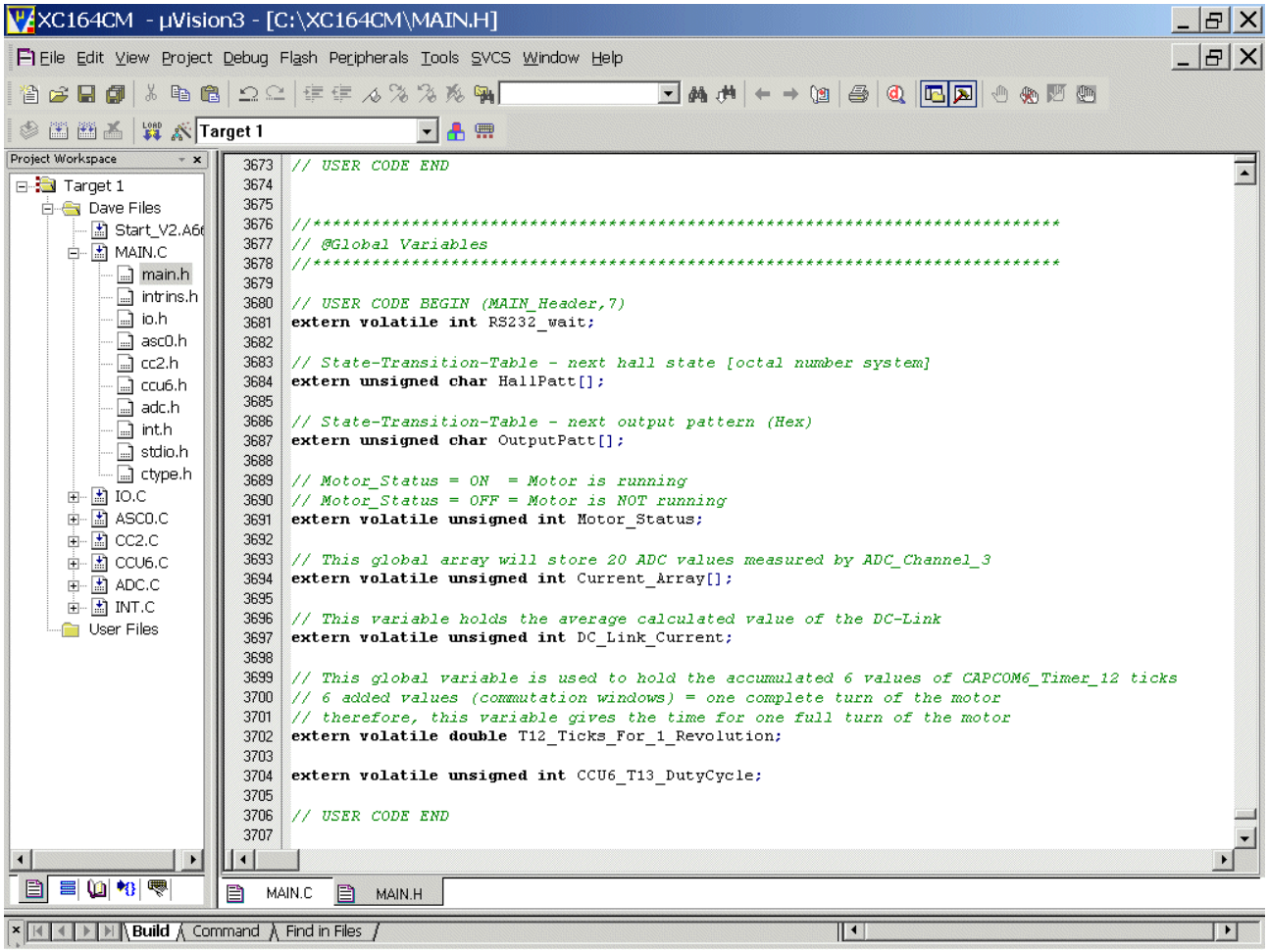


Double click main.h and change Declaration of Global Variables from:

```
// State-Transition-Table - next hall state [octal number system]
extern const unsigned char HallPatt[];

// State-Transition-Table - next output pattern (Hex)
extern const unsigned char OutputPatt[];
```

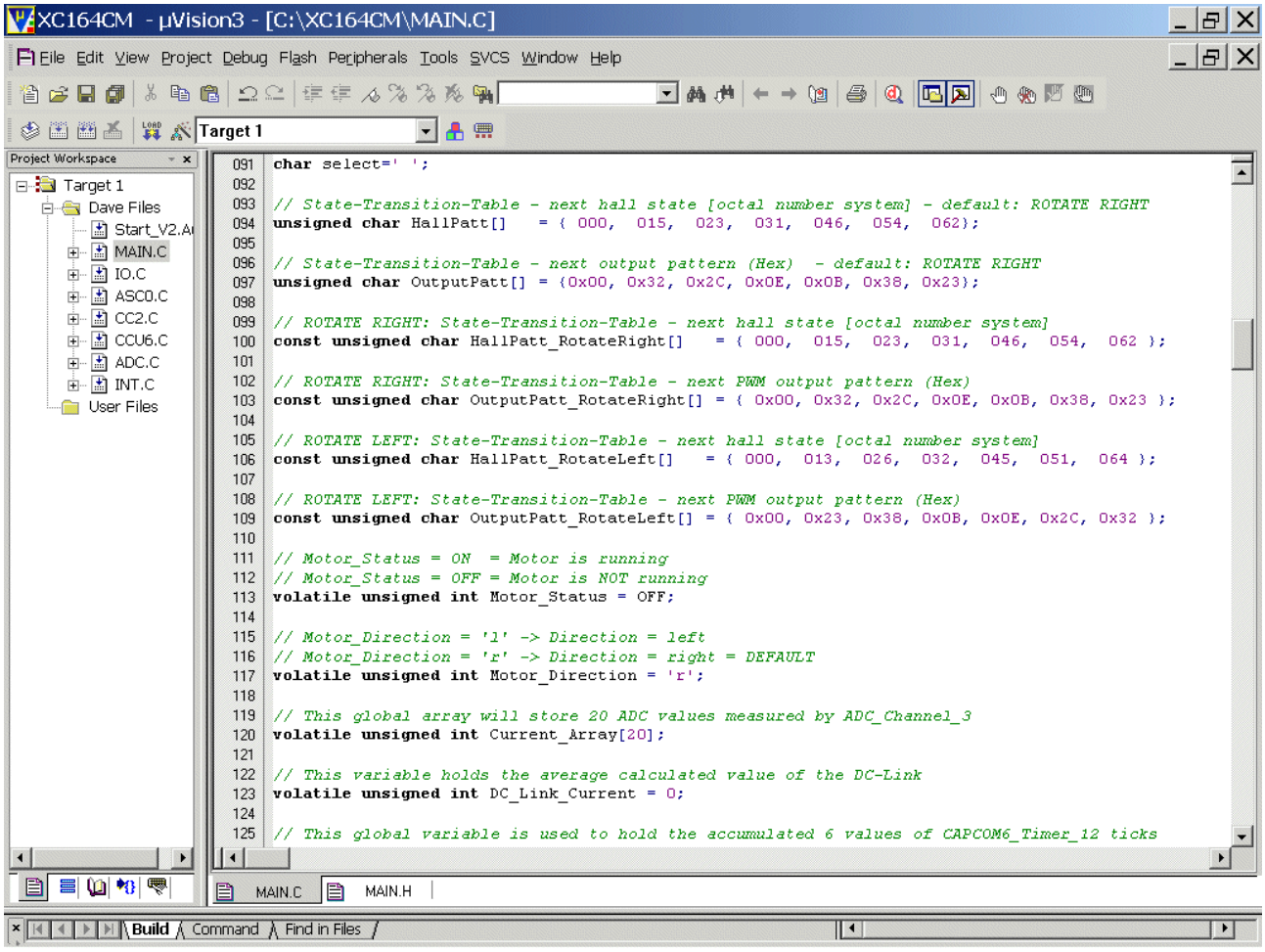
to:

```
// State-Transition-Table - next hall state [octal number system]
extern unsigned char HallPatt[];

// State-Transition-Table - next output pattern (Hex)
extern unsigned char OutputPatt[];
```



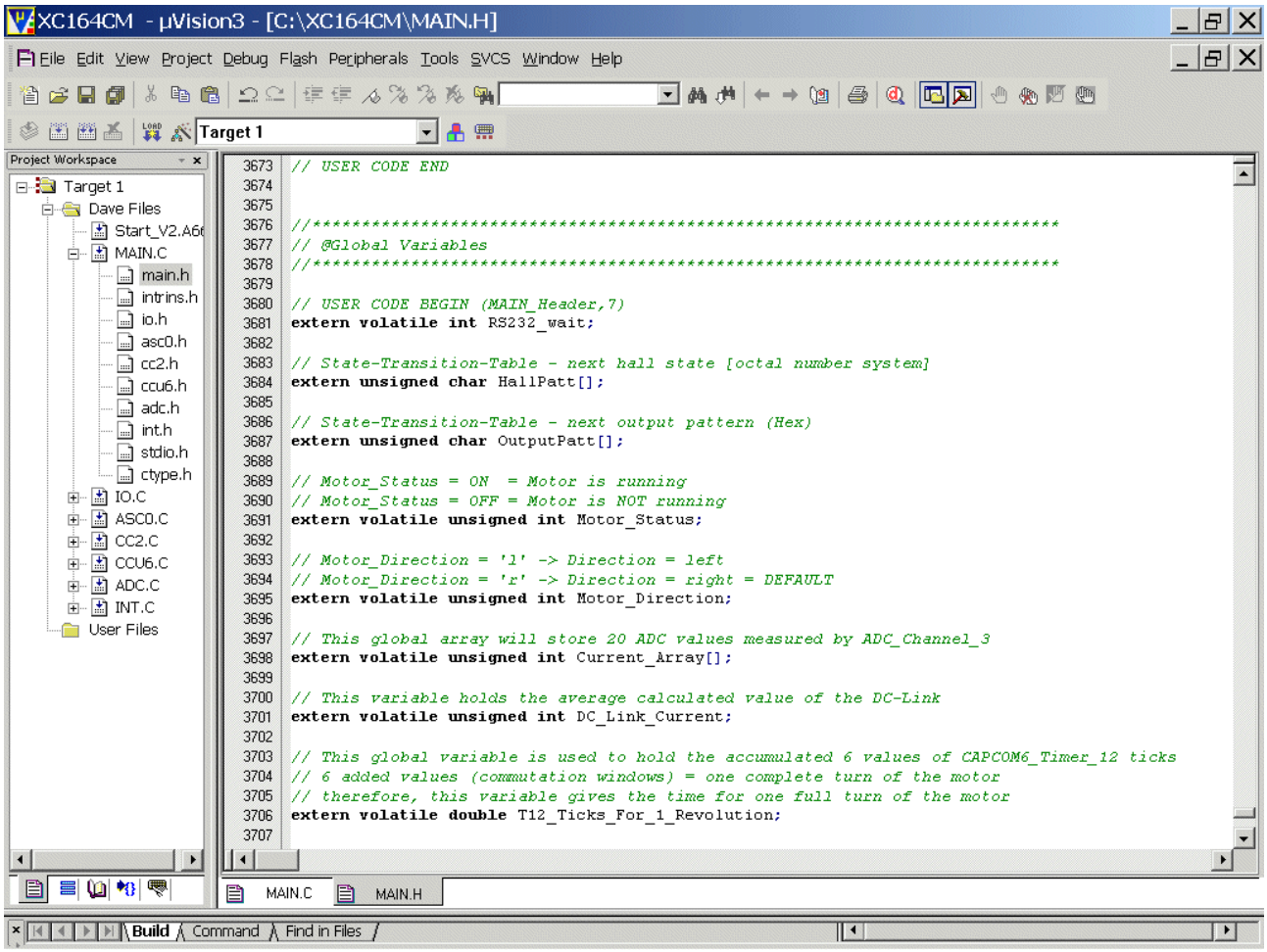
Double click MAIN.C and insert Definition of Global Variable:

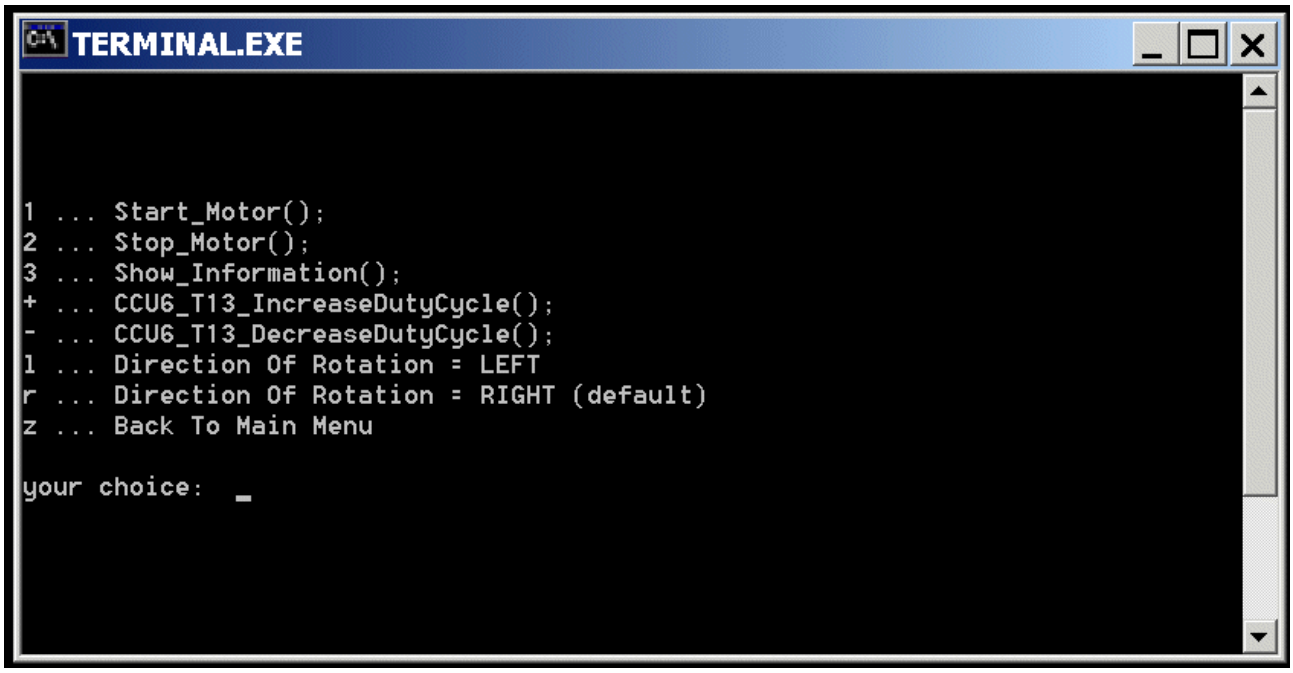
```
// Motor_Direction = 'l' -> Direction = left
// Motor_Direction = 'r' -> Direction = right = DEFAULT
volatile unsigned int Motor_Direction = 'r';
```



Double click main.h and insert Declaration of Global Variable:

```
// Motor_Direction = 'l' -> Direction = left
// Motor_Direction = 'r' -> Direction = right = DEFAULT
extern volatile unsigned int Motor_Direction;
```



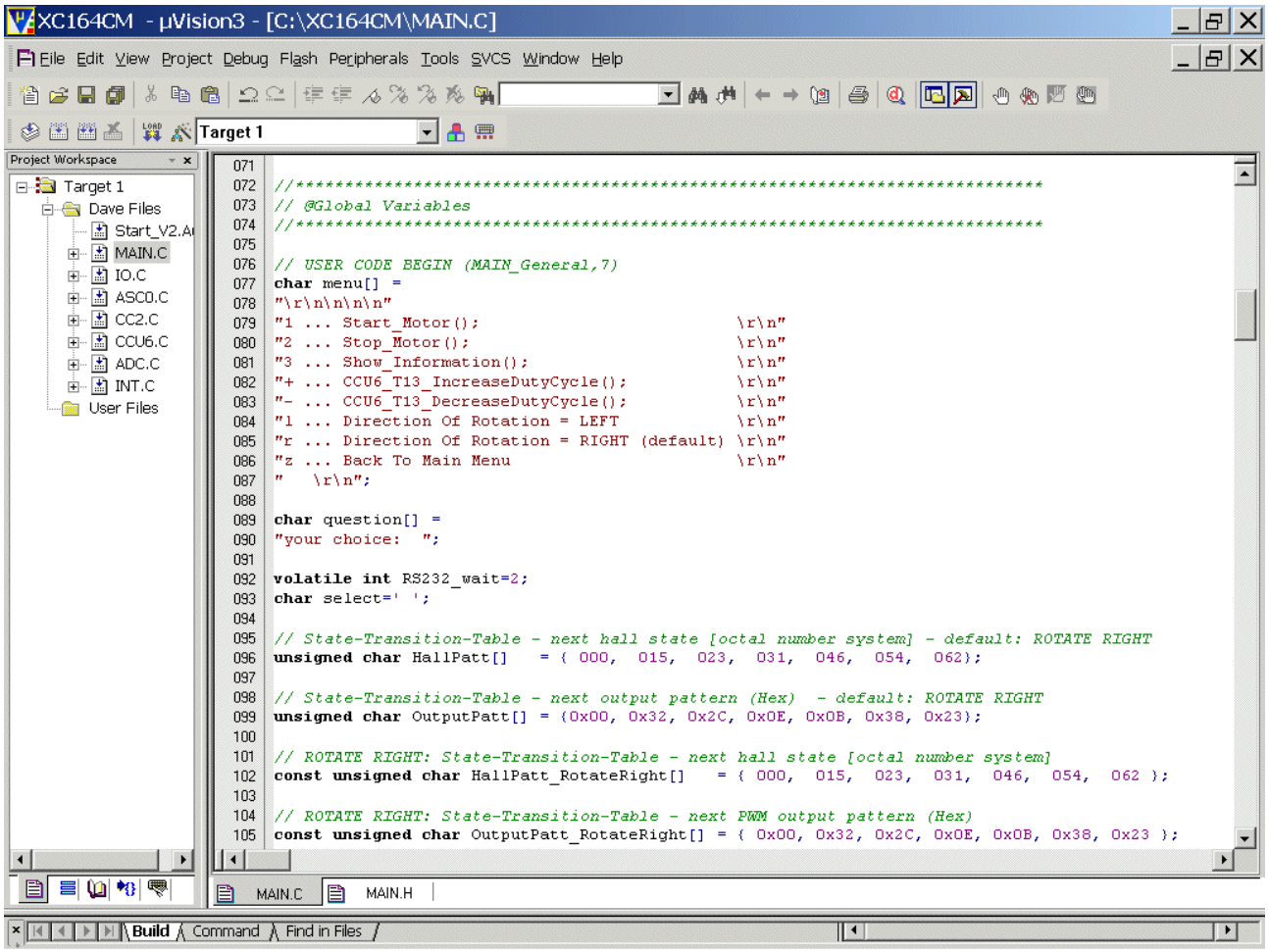


Double click MAIN.C and change Global Variable menu from:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor();                  \r\n"
"2 ... Stop_Motor();                   \r\n"
"3 ... Show_Information();              \r\n"
"+ ... CCU6_T13_IncreaseDutyCycle();  \r\n"
"- ... CCU6_T13_DecreaseDutyCycle();  \r\n"
"z ... Back To Main Menu               \r\n"
"   \r\n";
```

to:

```
char menu[] =
"\r\n\n\n\n"
"1 ... Start_Motor();                              \r\n"
"2 ... Stop_Motor();                               \r\n"
"3 ... Show_Information();                          \r\n"
"+ ... CCU6_T13_IncreaseDutyCycle();               \r\n"
"- ... CCU6_T13_DecreaseDutyCycle();               \r\n"
"l ... Direction Of Rotation = LEFT                \r\n"
"r ... Direction Of Rotation = RIGHT (default)     \r\n"
"z ... Back To Main Menu                           \r\n"
"   \r\n";
```

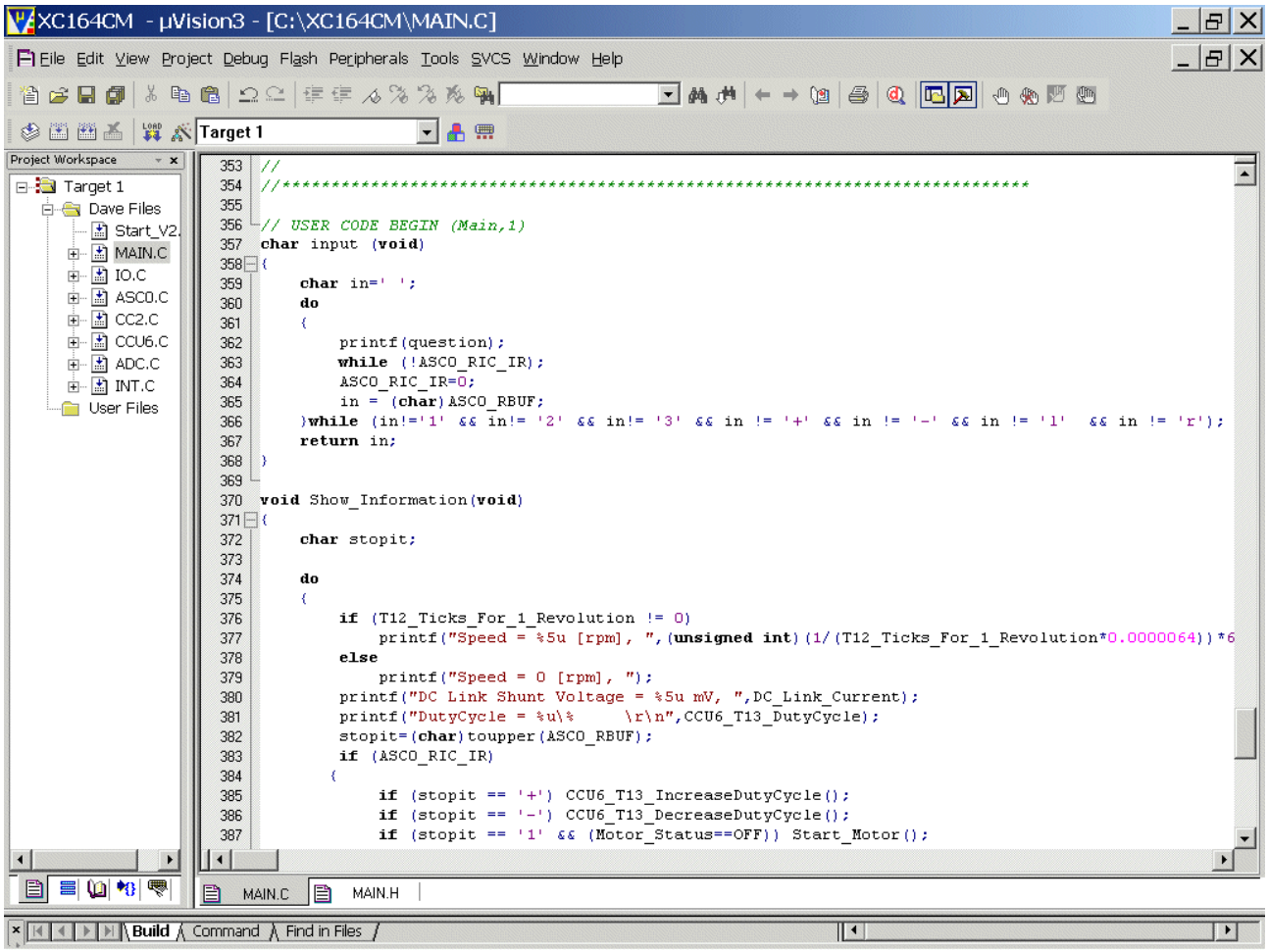

XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
071
072 //***
073 // @Global Variables
074 //***
075
076 // USER CODE BEGIN (MAIN_General,7)
077 char menu[] =
078 "\r\n\n\n\n"
079 "1 ... Start_Motor();   \r\n"
080 "2 ... Stop_Motor();   \r\n"
081 "3 ... Show_Information();   \r\n"
082 "+ ... CCU6_T13_IncreaseDutyCycle();   \r\n"
083 "- ... CCU6_T13_DecreaseDutyCycle();   \r\n"
084 "l ... Direction Of Rotation = LEFT   \r\n"
085 "r ... Direction Of Rotation = RIGHT (default) \r\n"
086 "z ... Back To Main Menu   \r\n"
087 "   \r\n";
088
089 char question[] =
090 "your choice: ";
091
092 volatile int RS232_wait=2;
093 char select=' ';
094
095 // State-Transition-Table - next hall state [octal number system] - default: ROTATE RIGHT
096 unsigned char HallPatt[]   = { 000, 015, 023, 031, 046, 054, 062};
097
098 // State-Transition-Table - next output pattern (Hex) - default: ROTATE RIGHT
099 unsigned char OutputPatt[] = {0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23};
100
101 // ROTATE RIGHT: State-Transition-Table - next hall state [octal number system]
102 const unsigned char HallPatt_RotateRight[]   = { 000, 015, 023, 031, 046, 054, 062 };
103
104 // ROTATE RIGHT: State-Transition-Table - next PWM output pattern (Hex)
105 const unsigned char OutputPatt_RotateRight[] = { 0x00, 0x32, 0x2C, 0x0E, 0x0B, 0x38, 0x23 };
MAIN.C
MAIN.H
Build
Command
Find in Files

Double click MAIN.C and change Fuction input from:

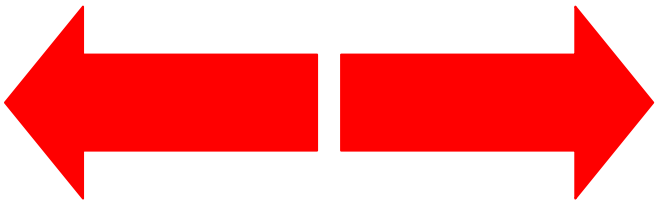
```
char input (void)
{
        char in=' ';
        do
        {
                printf(question);
                while (!ASC0_RIC_IR);
                ASC0_RIC_IR=0;
                in = (char)ASC0_RBUF;
        }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in != '-');
        return in;
}
```

to:

```
char input (void)
{
        char in=' ';
        do
        {
                printf(question);
                while (!ASC0_RIC_IR);
                ASC0_RIC_IR=0;
                in = (char)ASC0_RBUF;
        }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in != '-' && in != 'l'  && in != 'r');
        return in;
}
```

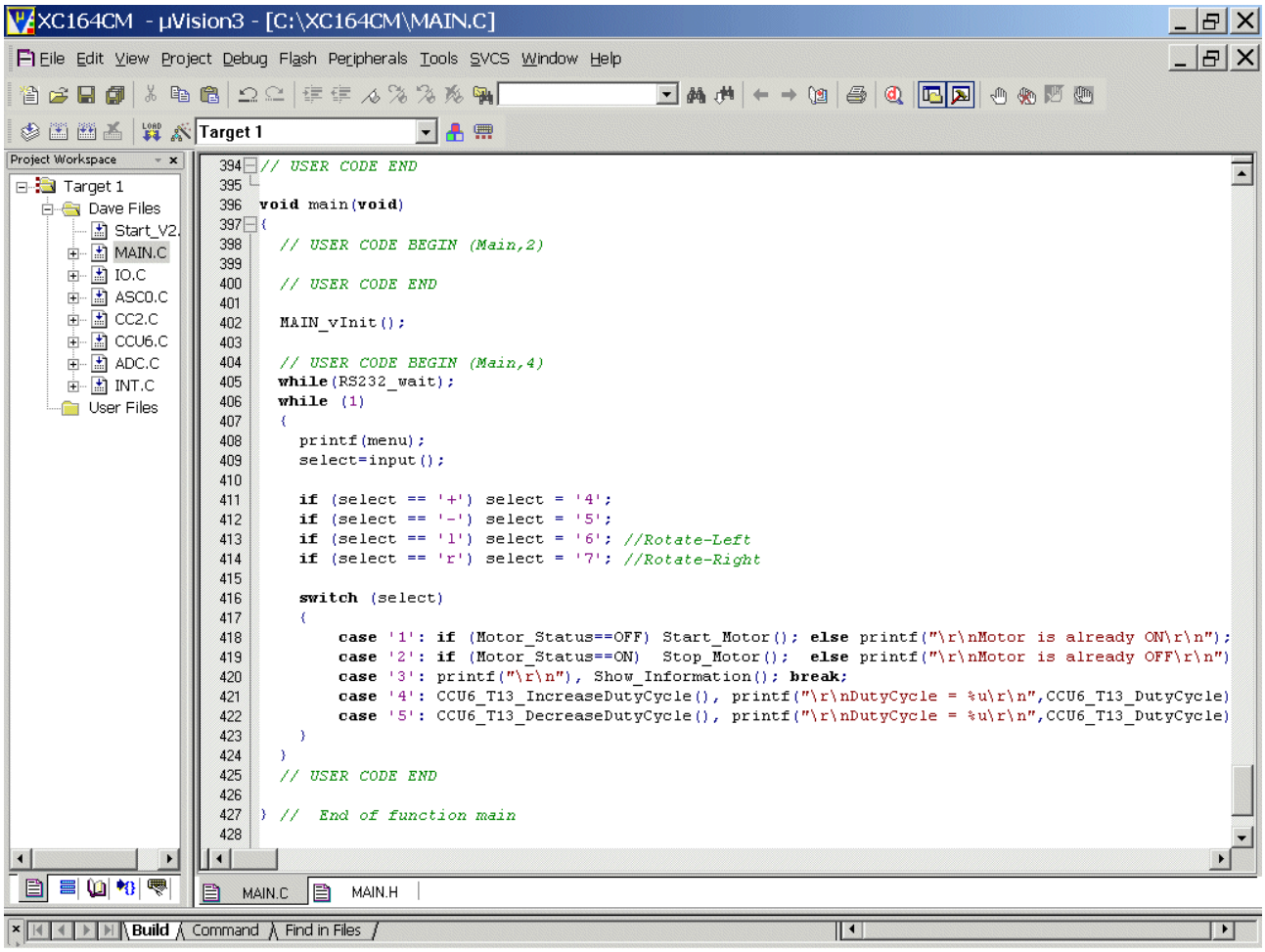



```
353 //
354 //*****************************************************************************
355
356 // USER CODE BEGIN (Main,1)
357 char input (void)
358 {
359     char in=' ';
360     do
361     {
362         printf(question);
363         while (!ASC0_RIC_IR);
364         ASC0_RIC_IR=0;
365         in = (char)ASC0_RBUF;
366     }while (in!='1' && in!= '2' && in!= '3' && in != '+' && in != '-' && in != 'l'  && in != 'r');
367     return in;
368 }
369
370 void Show_Information(void)
371 {
372     char stopit;
373
374     do
375     {
376         if (T12_Ticks_For_1_Revolution != 0)
377             printf("Speed = %5u [rpm], ",(unsigned int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*6
378         else
379             printf("Speed = 0 [rpm], ");
380         printf("DC Link Shunt Voltage = %5u mV, ",DC_Link_Current);
381         printf("DutyCycle = %u\%      \r\n",CCU6_T13_DutyCycle);
382         stopit=(char)toupper(ASC0_RBUF);
383         if (ASC0_RIC_IR)
384        {
385             if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
386             if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
387             if (stopit == '1' && (Motor_Status==OFF)) Start_Motor();
```



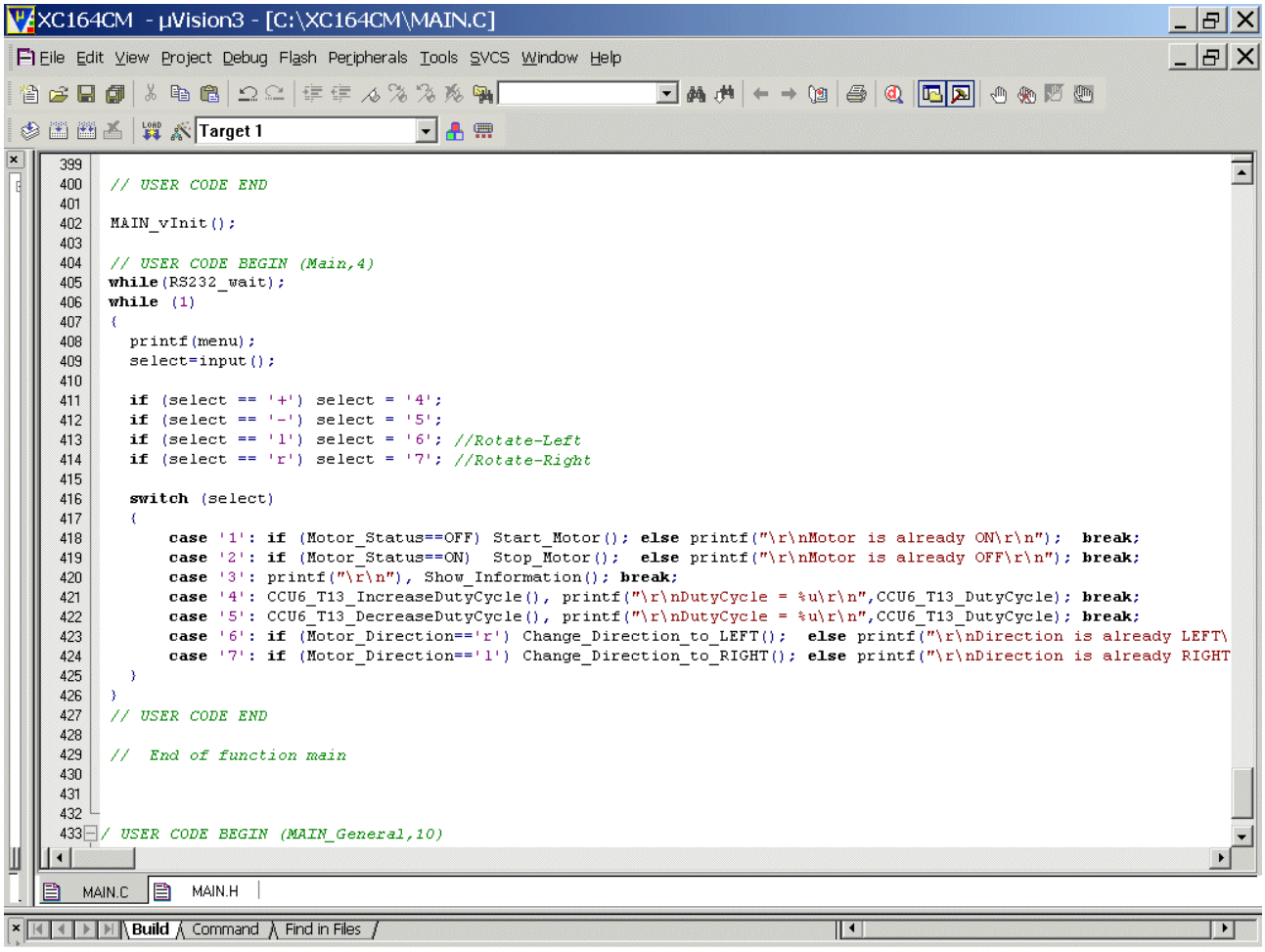
Double click MAIN.C and insert Code:

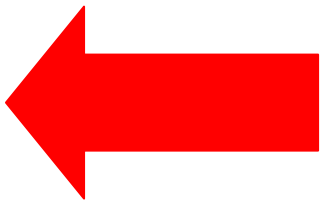
```
  if (select == 'l') select = '6'; //Rotate-Left
  if (select == 'r') select = '7'; //Rotate-Right
```



Double click MAIN.C and insert Code:

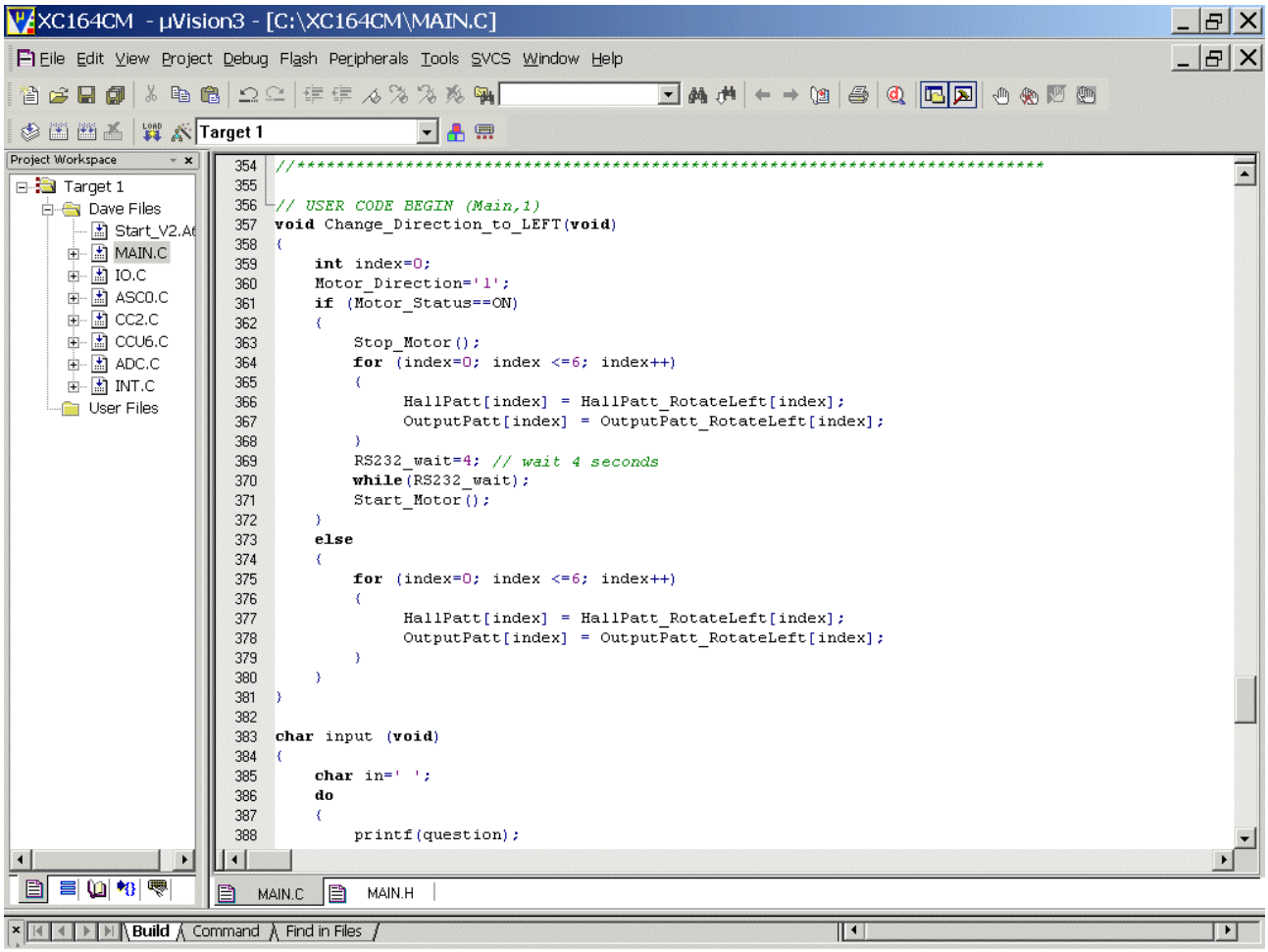
```
case '6': if (Motor_Direction=='r') Change_Direction_to_LEFT();  else printf("\r\nDirection is already LEFT\r\n");  break;
case '7': if (Motor_Direction=='l') Change_Direction_to_RIGHT(); else printf("\r\nDirection is already RIGHT\r\n"); break;
```

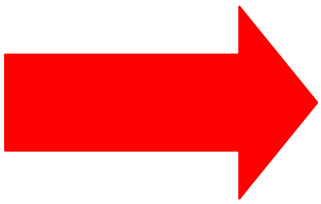




Double click MAIN.C and insert Code:

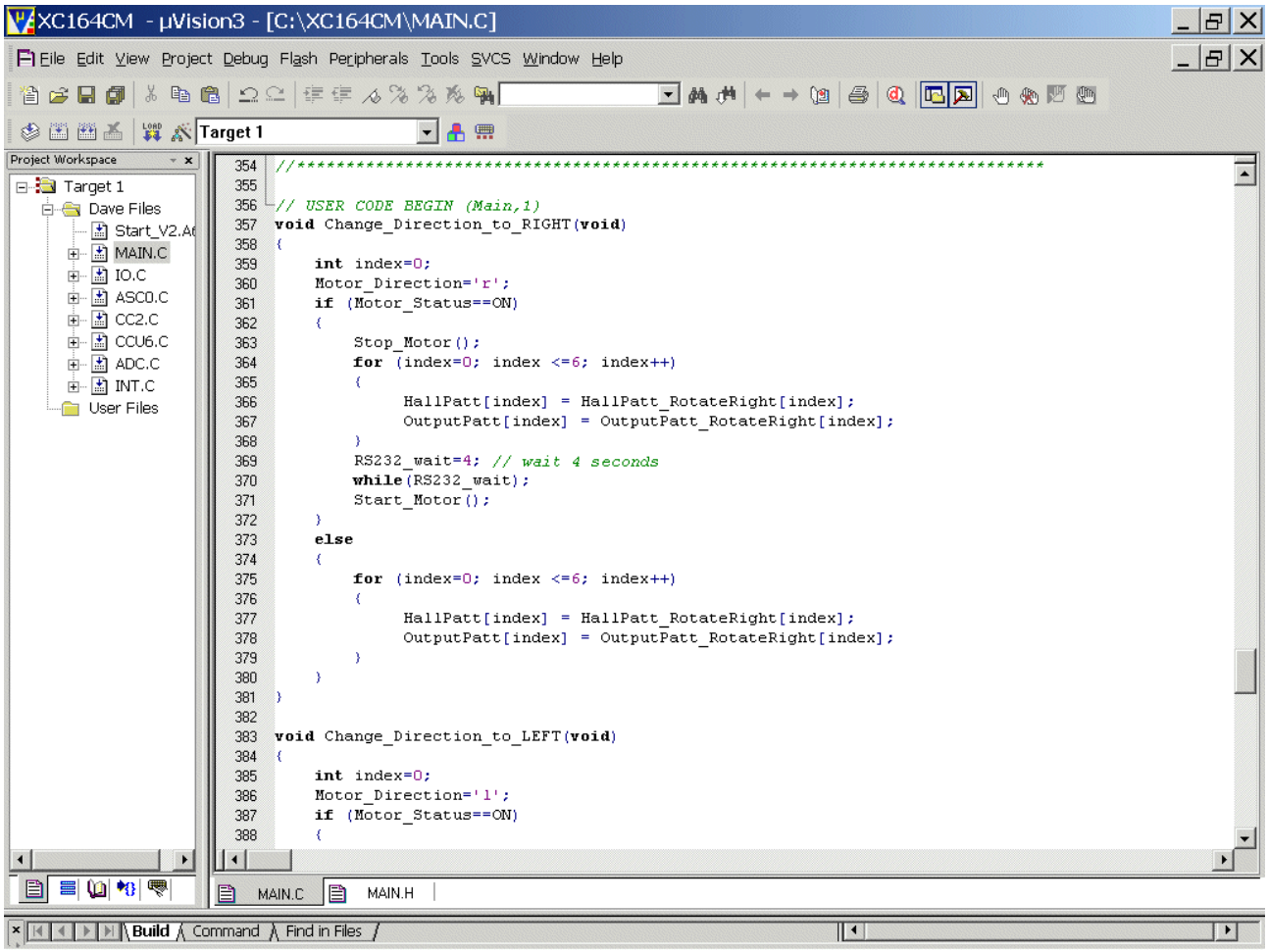
```
void Change_Direction_to_LEFT(void)
{
    int index=0;
    Motor_Direction='l';
    if (Motor_Status==ON)
    {
        Stop_Motor();
        for (index=0; index <=6; index++)
        {
            HallPatt[index] = HallPatt_RotateLeft[index];
            OutputPatt[index] = OutputPatt_RotateLeft[index];
        }
        RS232_wait=4; // wait 4 seconds
        while(RS232_wait);
        Start_Motor();
    }
    else
    {
         for (index=0; index <=6; index++)
        {
            HallPatt[index] = HallPatt_RotateLeft[index];
            OutputPatt[index] = OutputPatt_RotateLeft[index];
        }
    }
}
```


XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
354 //***
355
356 // USER CODE BEGIN (Main,1)
357 void Change_Direction_to_LEFT(void)
358 {
359 int index=0;
360 Motor_Direction='l';
361 if (Motor_Status==ON)
362 {
363 Stop_Motor();
364 for (index=0; index <=6; index++)
365 {
366 HallPatt[index] = HallPatt_RotateLeft[index];
367 OutputPatt[index] = OutputPatt_RotateLeft[index];
368 }
369 RS232_wait=4; // wait 4 seconds
370 while(RS232_wait);
371 Start_Motor();
372 }
373 else
374 {
375 for (index=0; index <=6; index++)
376 {
377 HallPatt[index] = HallPatt_RotateLeft[index];
378 OutputPatt[index] = OutputPatt_RotateLeft[index];
379 }
380 }
381 }
382
383 char input (void)
384 {
385 char in=' ';
386 do
387 {
388 printf(question);
MAIN.C
MAIN.H
Build
Command
Find in Files



Double click MAIN.C and insert Code:

```
void Change_Direction_to_RIGHT(void)
{
    int index=0;
     Motor_Direction='r';
    if (Motor_Status==ON)
    {
        Stop_Motor();
        for (index=0; index <=6; index++)
        {
             HallPatt[index] = HallPatt_RotateRight[index];
             OutputPatt[index] = OutputPatt_RotateRight[index];
        }
        RS232_wait=4; // wait 4 seconds
        while(RS232_wait);
        Start_Motor();
    }
    else
    {
         for (index=0; index <=6; index++)
        {
             HallPatt[index] = HallPatt_RotateRight[index];
             OutputPatt[index] = OutputPatt_RotateRight[index];
        }
    }
}
```

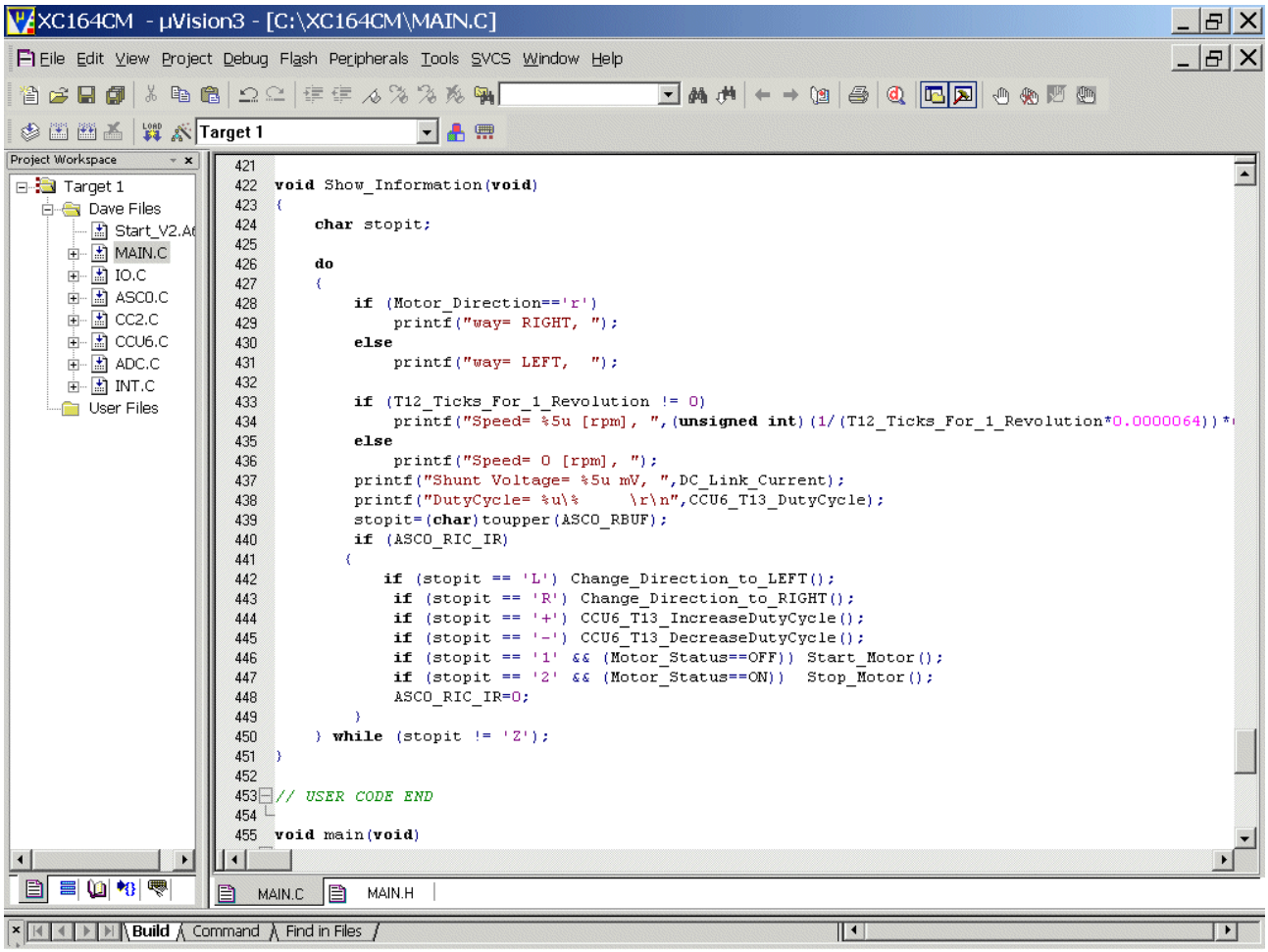

XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
354 //***
355
356 // USER CODE BEGIN (Main,1)
357 void Change_Direction_to_RIGHT(void)
358 {
359 int index=0;
360 Motor_Direction='r';
361 if (Motor_Status==ON)
362 {
363 Stop_Motor();
364 for (index=0; index <=6; index++)
365 {
366 HallPatt[index] = HallPatt_RotateRight[index];
367 OutputPatt[index] = OutputPatt_RotateRight[index];
368 }
369 RS232_wait=4; // wait 4 seconds
370 while(RS232_wait);
371 Start_Motor();
372 }
373 else
374 {
375 for (index=0; index <=6; index++)
376 {
377 HallPatt[index] = HallPatt_RotateRight[index];
378 OutputPatt[index] = OutputPatt_RotateRight[index];
379 }
380 }
381 }
382
383 void Change_Direction_to_LEFT(void)
384 {
385 int index=0;
386 Motor_Direction='l';
387 if (Motor_Status==ON)
388 {
MAIN.C
MAIN.H
Build
Command
Find in Files

Double click MAIN.C and change Code from:

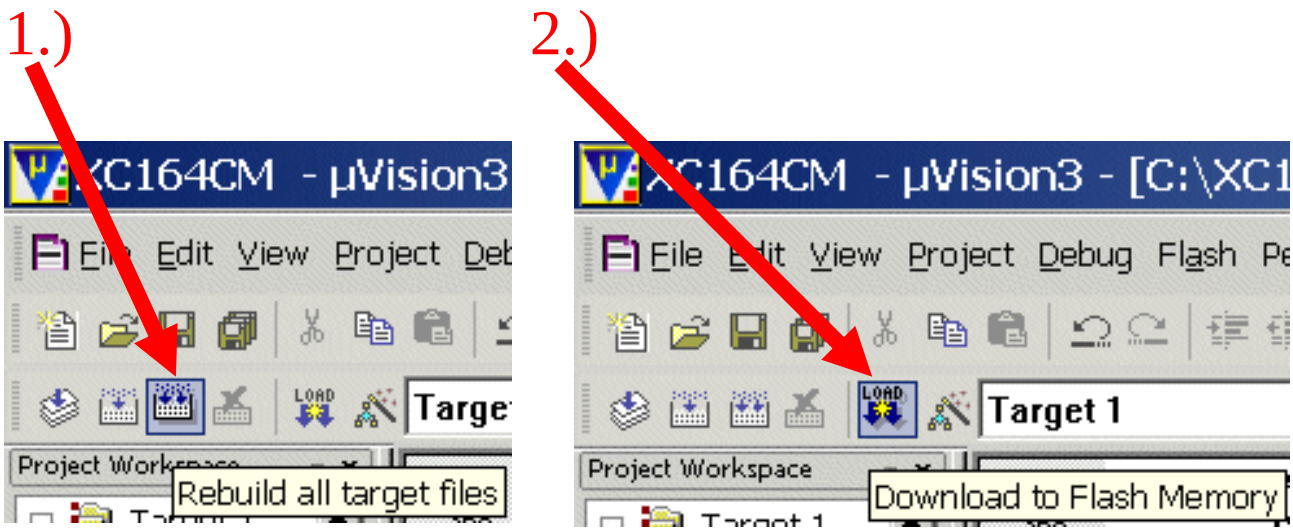
```
void Show_Information(void)
{
    char stopit;

    do
    {
        if (T12_Ticks_For_1_Revolution != 0)
            printf("Speed = %5u [rpm], ",(unsigned
int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*60);
        else
            printf("Speed = 0 [rpm], ");
        printf("DC Link Shunt Voltage = %5u mV,
",DC_Link_Current);
        printf("DutyCycle = %u\%     \r\n",CCU6_T13_DutyCycle);
        stopit=(char)toupper(ASC0_RBUF);
        if (ASC0_RIC_IR)
        {
            if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
            if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
            if (stopit == '1' && (Motor_Status==OFF))
Start_Motor();
            if (stopit == '2' && (Motor_Status==ON))
Stop_Motor();
            ASC0_RIC_IR=0;
        }
    } while (stopit != 'Z');
}
```

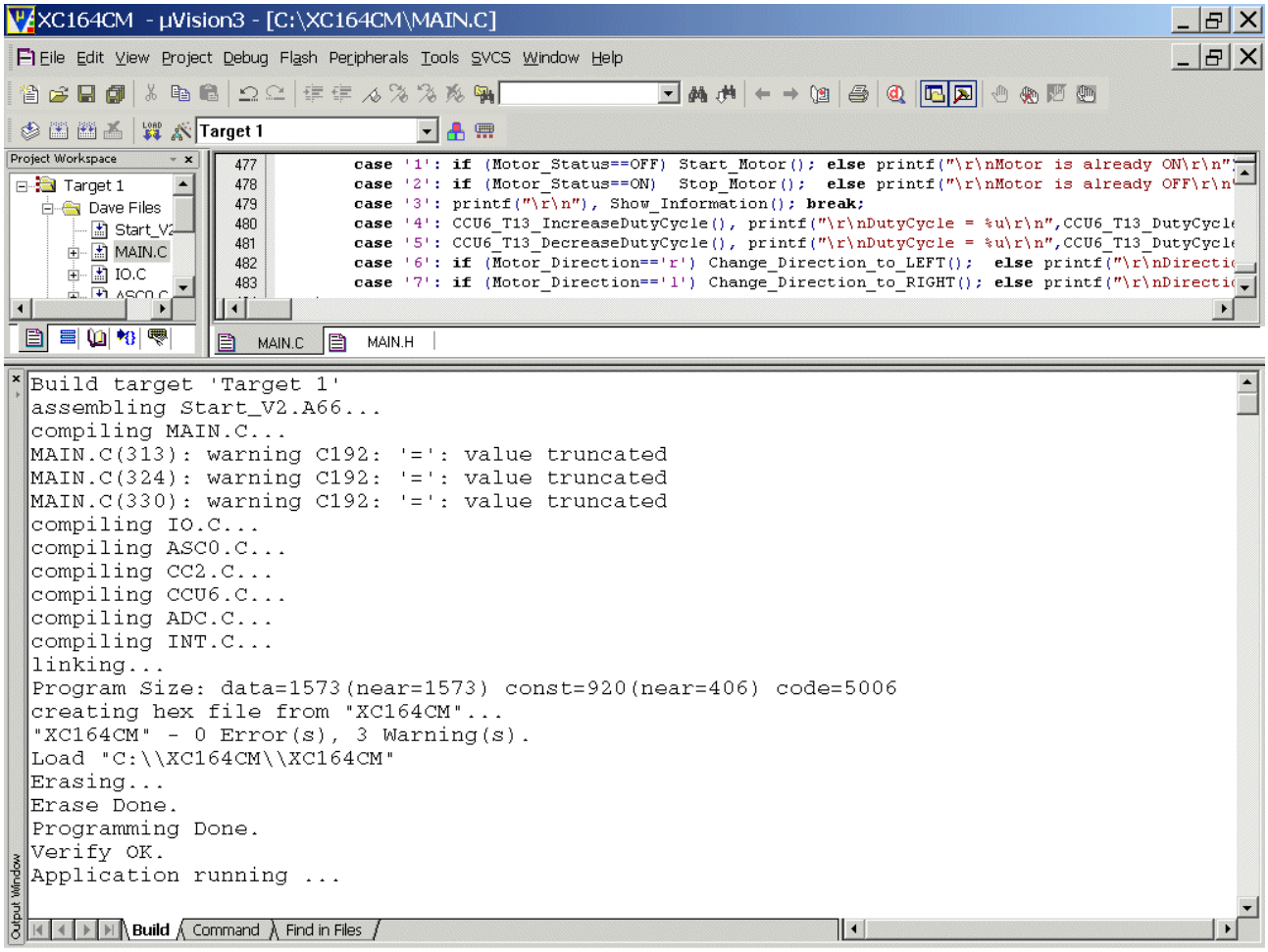
to:

```
void Show_Information(void)
{
    char stopit;

    do
    {
         if (Motor_Direction=='r')
            printf("way= RIGHT, ");
        else
            printf("way= LEFT,  ");

        if (T12_Ticks_For_1_Revolution != 0)
            printf("Speed= %5u [rpm], ",(unsigned
int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*60);
        else
            printf("Speed= 0 [rpm], ");
        printf("Shunt Voltage= %5u mV, ",DC_Link_Current);
        printf("DutyCycle= %u\%     \r\n",CCU6_T13_DutyCycle);
        stopit=(char)toupper(ASC0_RBUF);
        if (ASC0_RIC_IR)
        {
            if (stopit == 'L') Change_Direction_to_LEFT();
            if (stopit == 'R') Change_Direction_to_RIGHT();
            if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
            if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
            if (stopit == '1' && (Motor_Status==OFF))
Start_Motor();
            if (stopit == '2' && (Motor_Status==ON))
Stop_Motor();
            ASC0_RIC_IR=0;
        }
    } while (stopit != 'Z');
}
```

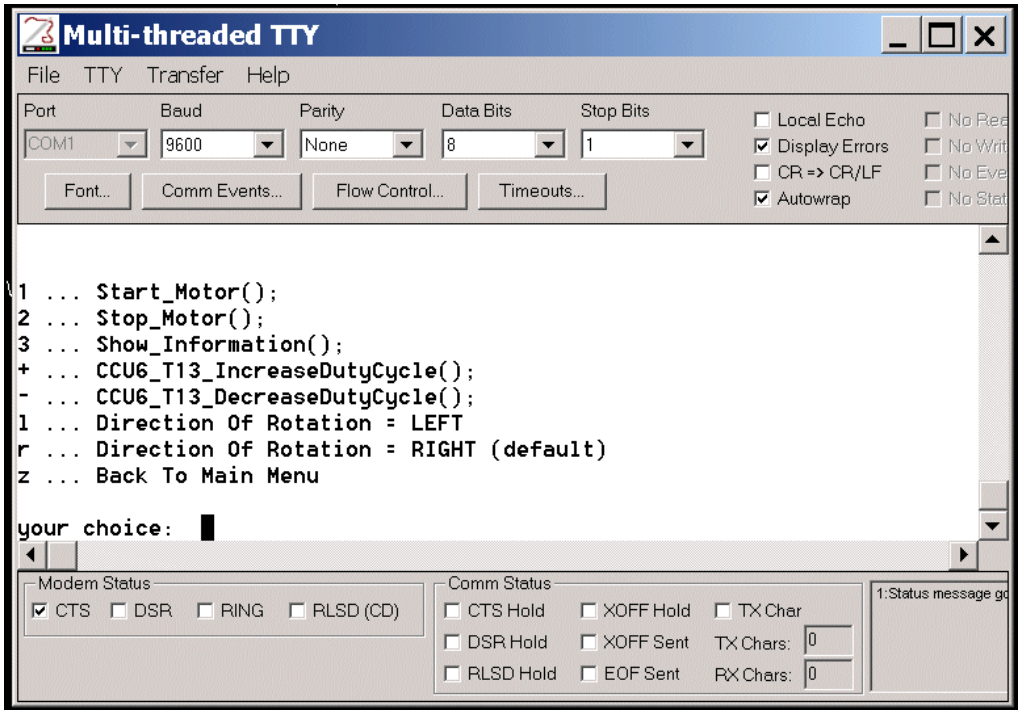

XC164CM - µVision3 - [C:\XC164CM\MAIN.C]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Dave Files
Start_V2.A
MAIN.C
IO.C
ASC0.C
CC2.C
CCU6.C
ADC.C
INT.C
User Files
421
422 void Show_Information(void)
423 {
424 char stopit;
425
426 do
427 {
428 if (Motor_Direction=='r')
429 printf("way= RIGHT, ");
430 else
431 printf("way= LEFT, ");
432
433 if (T12_Ticks_For_1_Revolution != 0)
434 printf("Speed= %5u [rpm], ",(unsigned int)(1/(T12_Ticks_For_1_Revolution*0.0000064))*
435 else
436 printf("Speed= 0 [rpm], ");
437 printf("Shunt Voltage= %5u mV, ",DC_Link_Current);
438 printf("DutyCycle= %u\% \r\n",CCU6_T13_DutyCycle);
439 stopit=(char)toupper(ASC0_RBUF);
440 if (ASC0_RIC_IR)
441 {
442 if (stopit == 'L') Change_Direction_to_LEFT();
443 if (stopit == 'R') Change_Direction_to_RIGHT();
444 if (stopit == '+') CCU6_T13_IncreaseDutyCycle();
445 if (stopit == '-') CCU6_T13_DecreaseDutyCycle();
446 if (stopit == '1' && (Motor_Status==OFF)) Start_Motor();
447 if (stopit == '2' && (Motor_Status==ON)) Stop_Motor();
448 ASC0_RIC_IR=0;
449 }
450 } while (stopit != 'Z');
451 }
452
453 // USER CODE END
454
455 void main(void)
MAIN.C
MAIN.H
Build
Command
Find in Files

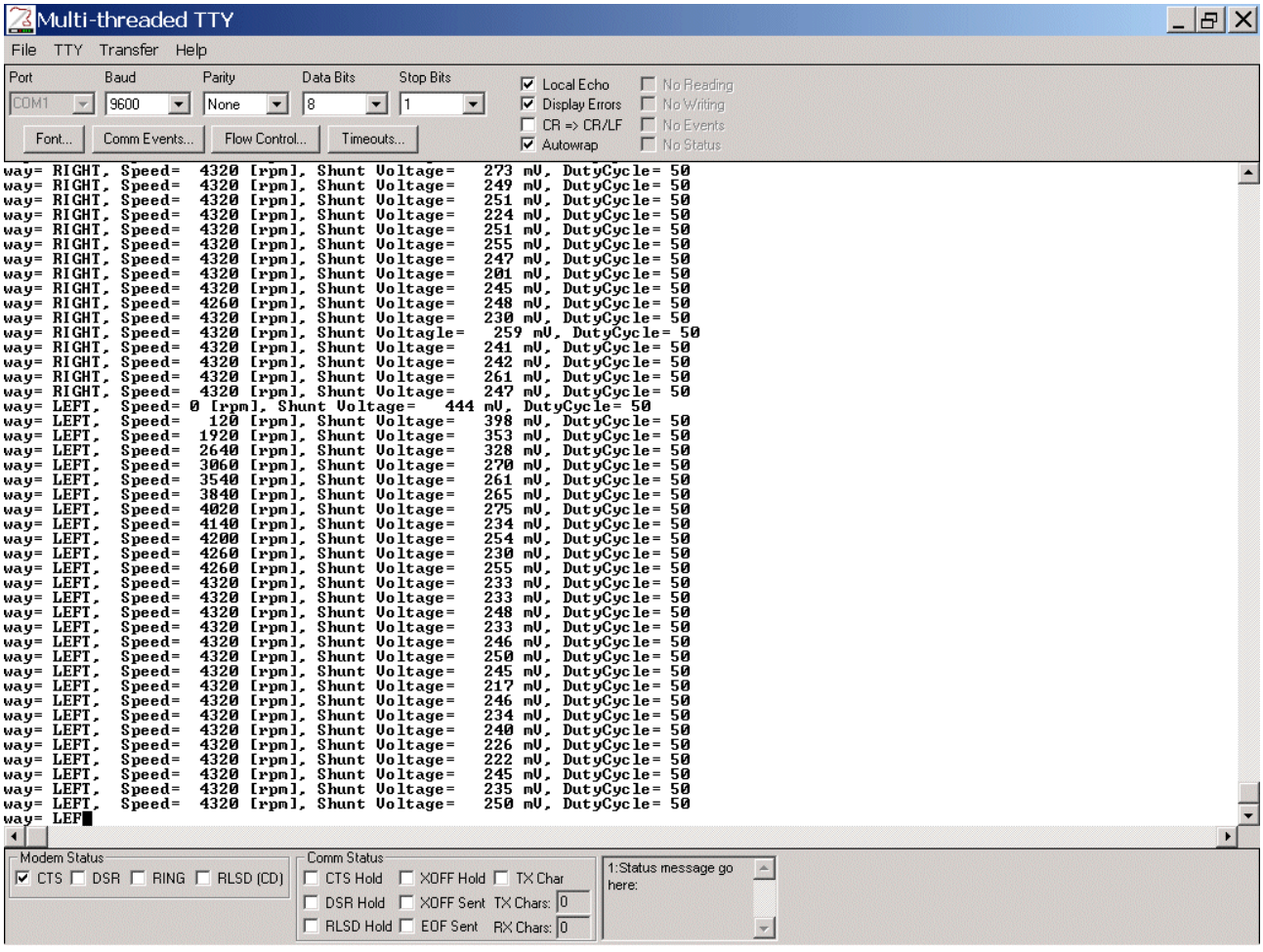
Build Application:





And have fun:


Multi-threaded TTY
File TTY Transfer Help
Port Baud Parity Data Bits Stop Bits
COM1 9600 None 8 1
Local Echo
Display Errors
CR => CR/LF
Autowrap
Font... Comm Events... Flow Control... Timeouts...
1 ... Start_Motor();
2 ... Stop_Motor();
3 ... Show_Information();
+ ... CCU6_T13_IncreaseDutyCycle();
- ... CCU6_T13_DecreaseDutyCycle();
l ... Direction Of Rotation = LEFT
r ... Direction Of Rotation = RIGHT (default)
z ... Back To Main Menu
your choice:
Modem Status
CTS DSR RING RLSD (CD)
Comm Status
CTS Hold XOFF Hold TX Char
DSR Hold XOFF Sent TX Chars:
RLSD Hold EOF Sent RX Chars:

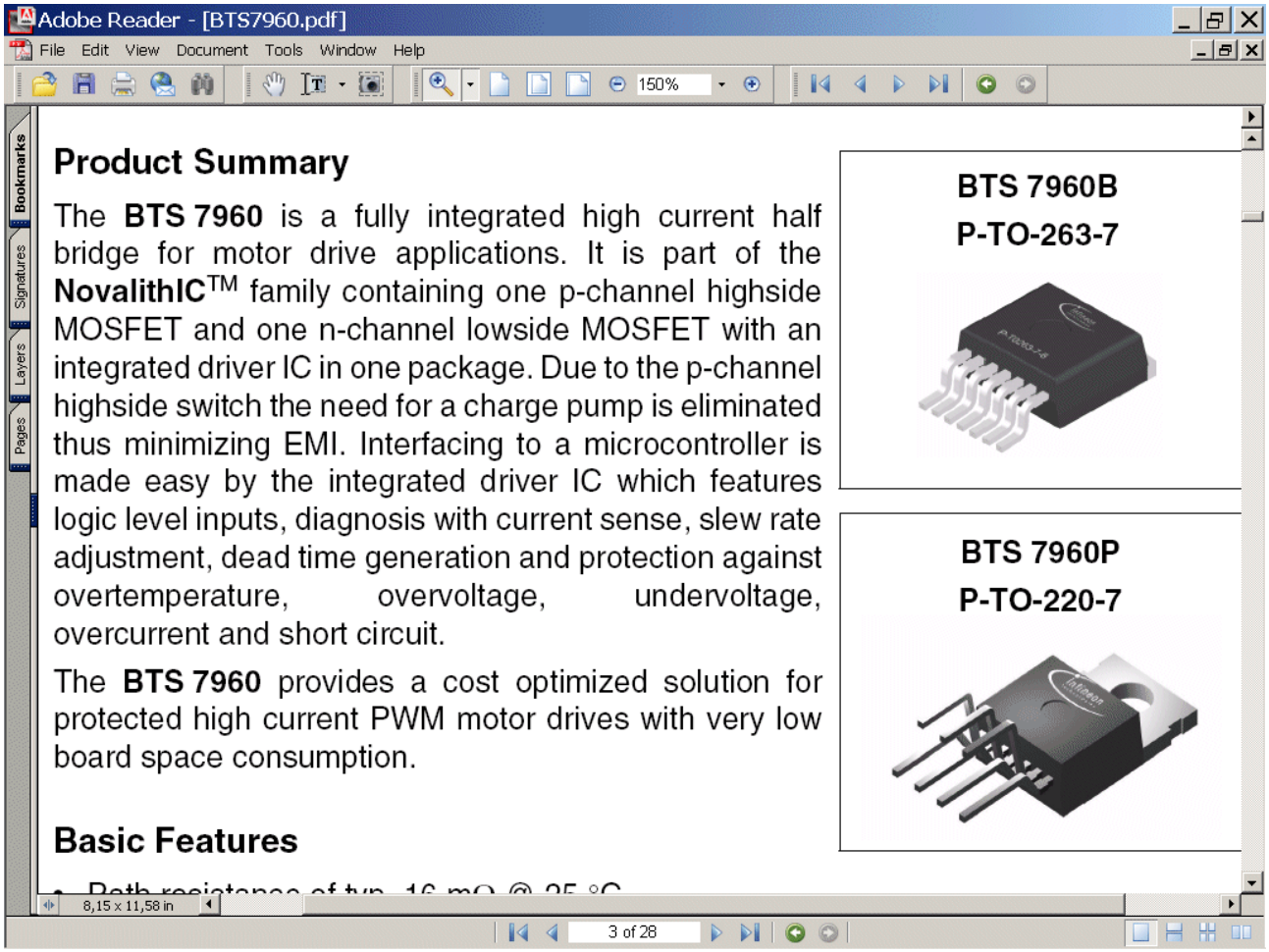

Multi-threaded TTY
File TTY Transfer Help
Port Baud Parity Data Bits Stop Bits
COM1 9600 None 8 1
Local Echo
Display Errors
CR => CR/LF
Autowrap
Font... Comm Events... Flow Control... Timeouts...
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 273 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 249 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 251 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 224 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 251 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 255 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 247 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 201 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 245 mV, DutyCycle= 50
way= RIGHT, Speed= 4260 [rpm], Shunt Voltage= 248 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 230 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltagle= 259 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 241 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 242 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 261 mV, DutyCycle= 50
way= RIGHT, Speed= 4320 [rpm], Shunt Voltage= 247 mV, DutyCycle= 50
way= LEFT, Speed= 0 [rpm], Shunt Voltage= 444 mV, DutyCycle= 50
way= LEFT, Speed= 120 [rpm], Shunt Voltage= 398 mV, DutyCycle= 50
way= LEFT, Speed= 1920 [rpm], Shunt Voltage= 353 mV, DutyCycle= 50
way= LEFT, Speed= 2640 [rpm], Shunt Voltage= 328 mV, DutyCycle= 50
way= LEFT, Speed= 3060 [rpm], Shunt Voltage= 270 mV, DutyCycle= 50
way= LEFT, Speed= 3540 [rpm], Shunt Voltage= 261 mV, DutyCycle= 50
way= LEFT, Speed= 3840 [rpm], Shunt Voltage= 265 mV, DutyCycle= 50
way= LEFT, Speed= 4020 [rpm], Shunt Voltage= 275 mV, DutyCycle= 50
way= LEFT, Speed= 4140 [rpm], Shunt Voltage= 234 mV, DutyCycle= 50
way= LEFT, Speed= 4200 [rpm], Shunt Voltage= 254 mV, DutyCycle= 50
way= LEFT, Speed= 4260 [rpm], Shunt Voltage= 230 mV, DutyCycle= 50
way= LEFT, Speed= 4260 [rpm], Shunt Voltage= 255 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 233 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 233 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 248 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 233 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 246 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 250 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 245 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 217 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 246 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 234 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 240 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 226 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 222 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 245 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 235 mV, DutyCycle= 50
way= LEFT, Speed= 4320 [rpm], Shunt Voltage= 250 mV, DutyCycle= 50
way= LEF
Modem Status
CTS DSR RING RLSD (CD)
Comm Status
CTS Hold XOFF Hold TX Char
DSR Hold XOFF Sent TX Chars:
RLSD Hold EOF Sent RX Chars:
1:Status message go here:

11.)

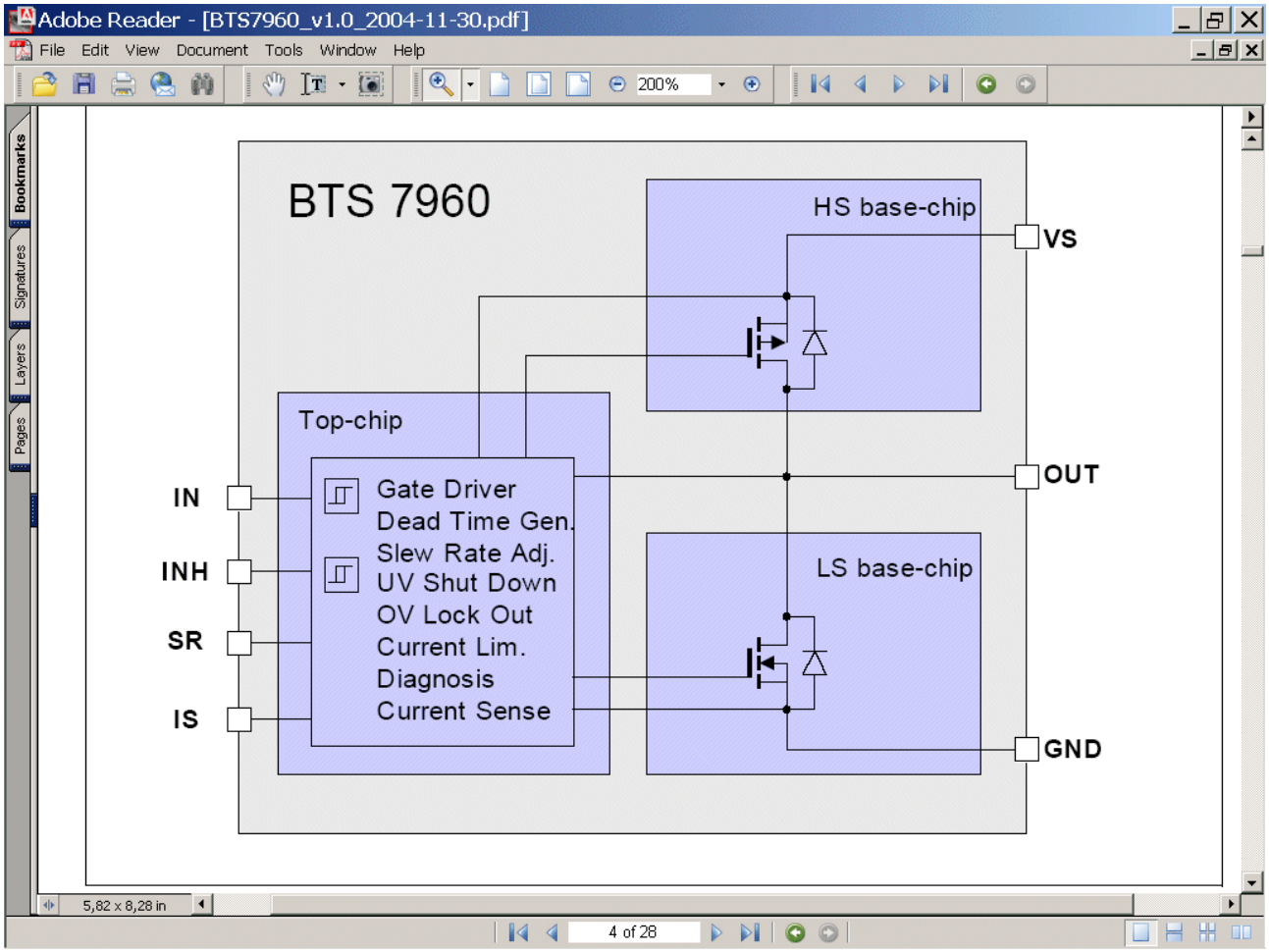
A Spot Of Information About:

Using three
BTS 7960
[high current half bridge (for motor drive applications) containing one highside and one lowside switch with driver]
as a B6 Bridge:

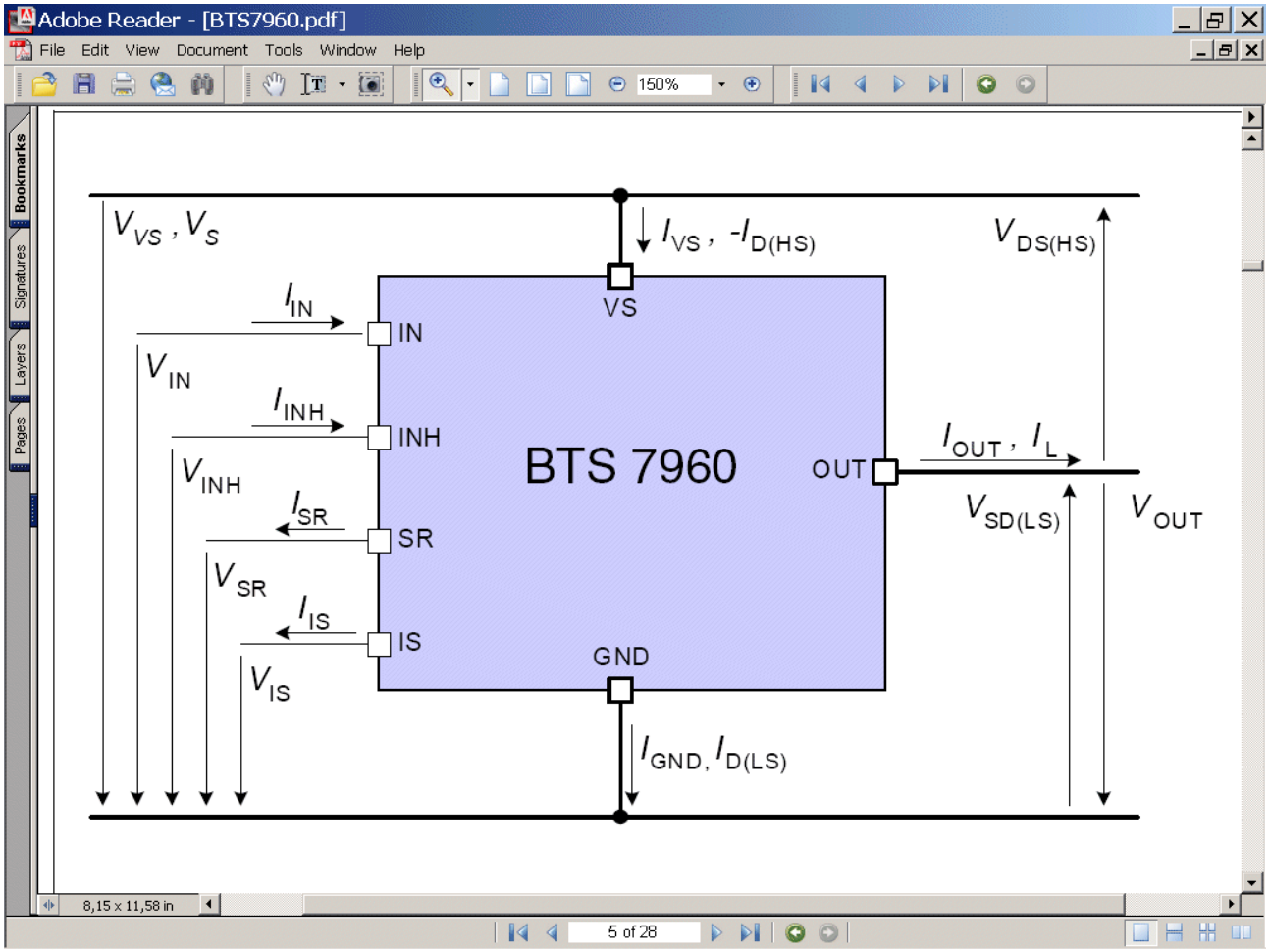
Product Summary:



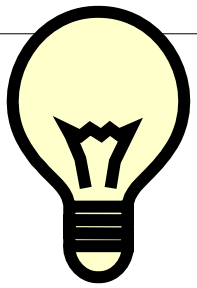
Block Diagram:



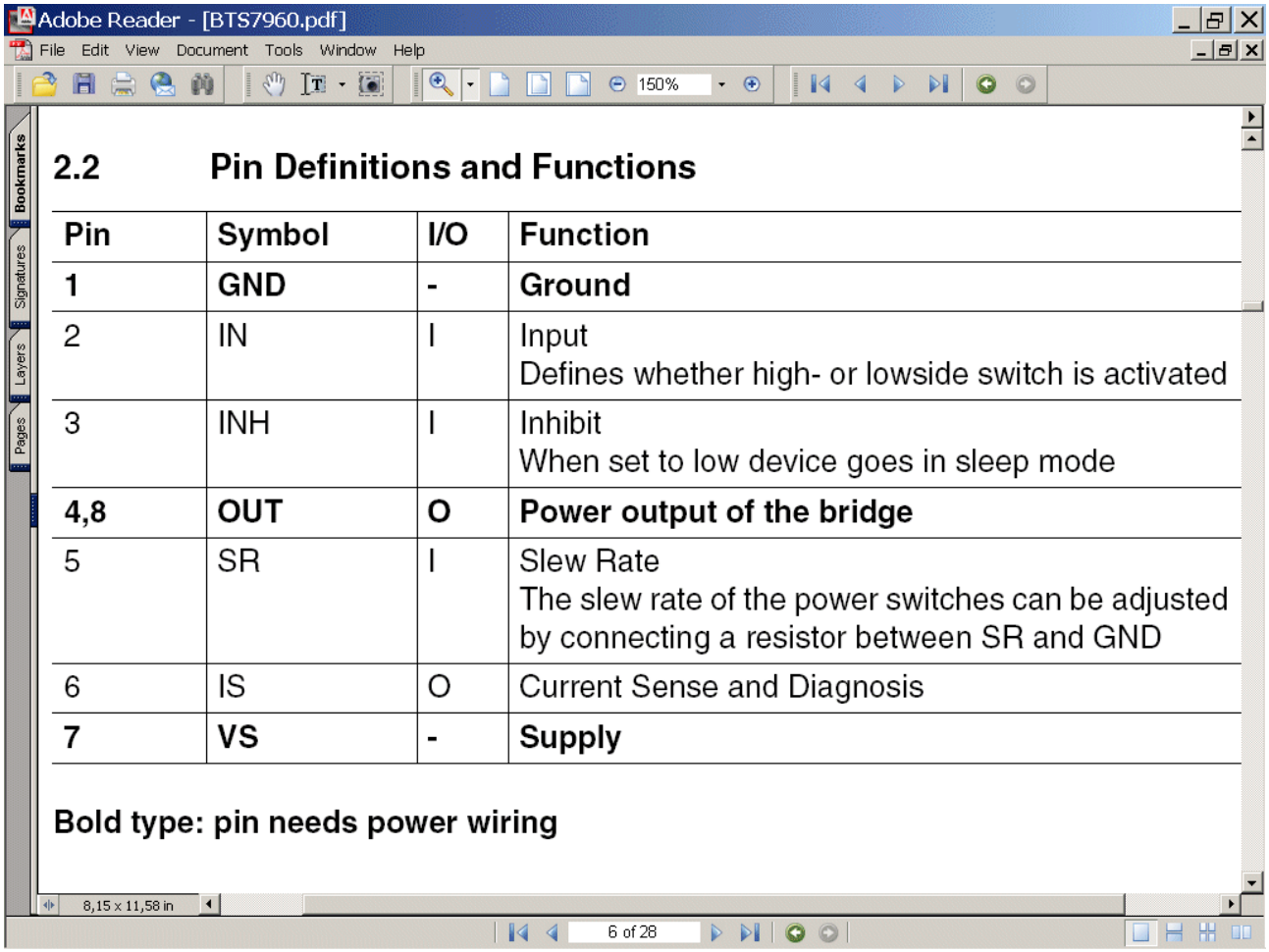
Terms:



CAPCOM6 Outputs	BTS 7960 #1 (Inputs)	BTS 7960 #2 (Inputs)	BTS 7960 #3 (Inputs)
CC60	IN (high/low-activated)		
COUT60	INH (Inhibit)		
CC61		IN (high/low-activated)	
COUT61		INH (Inhibit)	
CC62			IN (high/low-activated)
COUT62			INH (Inhibit)



Pin Definitions and Functions:


Adobe Reader - [BTS7960.pdf]

File Edit View Document Tools Window Help

Bookmarks Signatures Layers Pages

2.2 Pin Definitions and Functions

Pin	Symbol	I/O	Function
1	**GND**	**-**	**Ground**
2	IN	I	Input Defines whether high- or lowside switch is activated
3	INH	I	Inhibit When set to low device goes in sleep mode
4,8	**OUT**	**O**	**Power output of the bridge**
5	SR	I	Slew Rate The slew rate of the power switches can be adjusted by connecting a resistor between SR and GND
6	IS	O	Current Sense and Diagnosis
7	**VS**	**-**	**Supply**

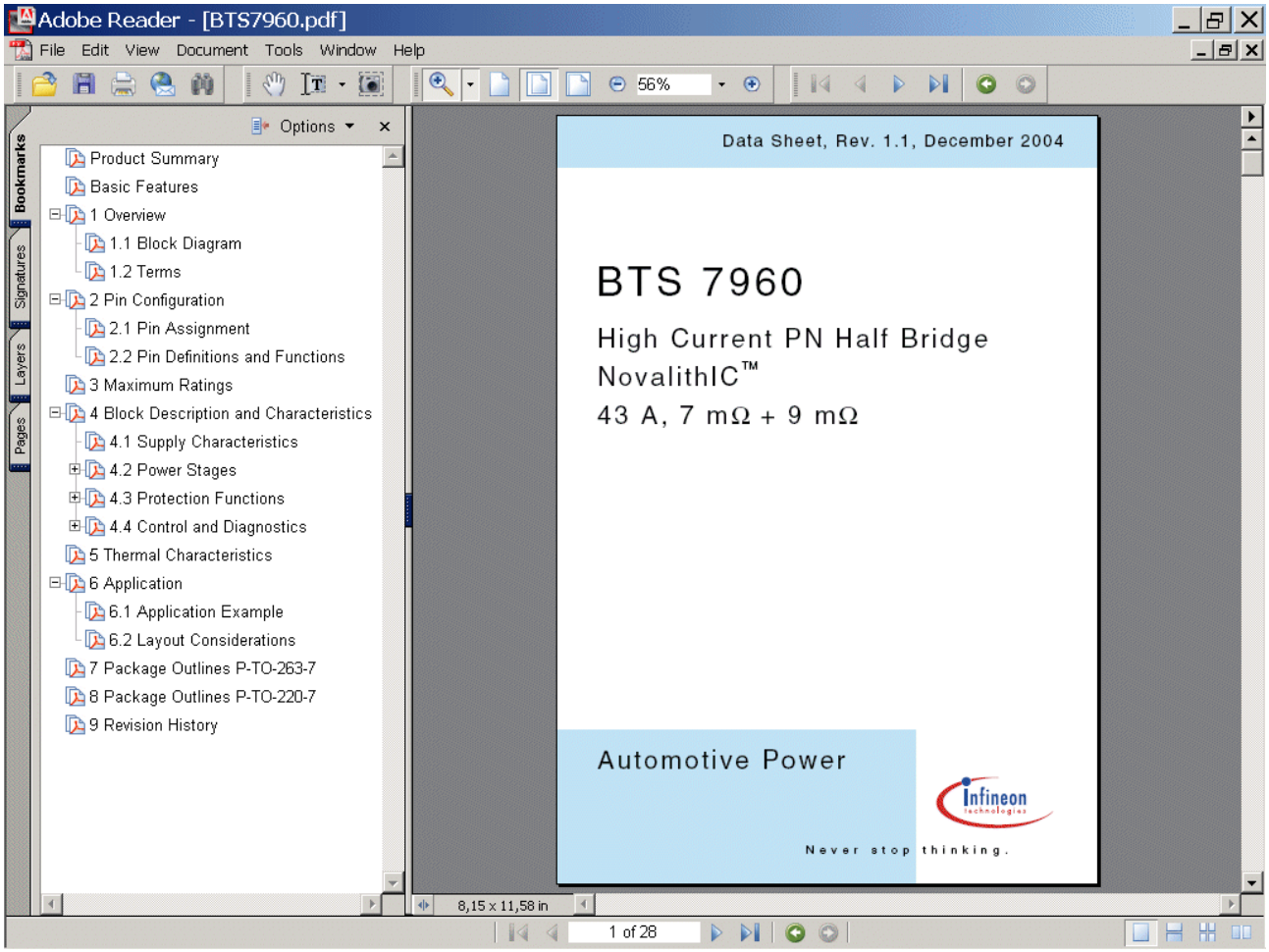
Bold type: pin needs power wiring

8,15 x 11,58 in

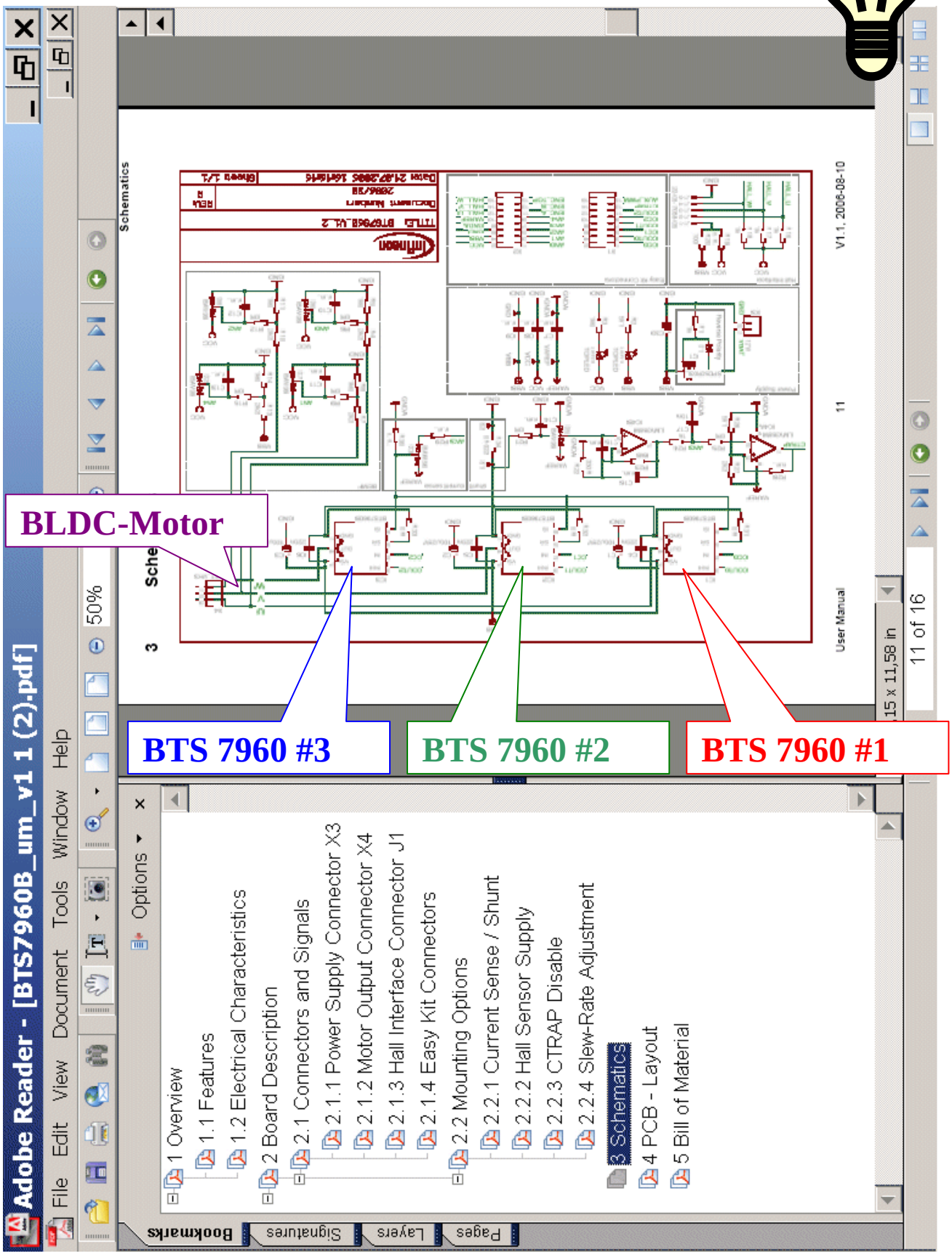
6 of 28

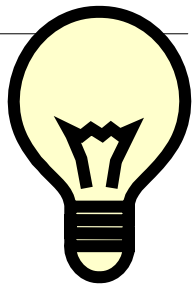
CAPCOM6 Outputs	BTS 7960 #1 (Inputs)	BTS 7960 #2 (Inputs)	BTS 7960 #3 (Inputs)
CC60	IN (high/low-activated)		
COUT60	INH (Inhibit)		
CC61		IN (high/low-activated)	
COUT61		INH (Inhibit)	
CC62			IN (high/low-activated)
COUT62			INH (Inhibit)

Note:
For further information, please refer to the BTS 7960 Data Sheet:



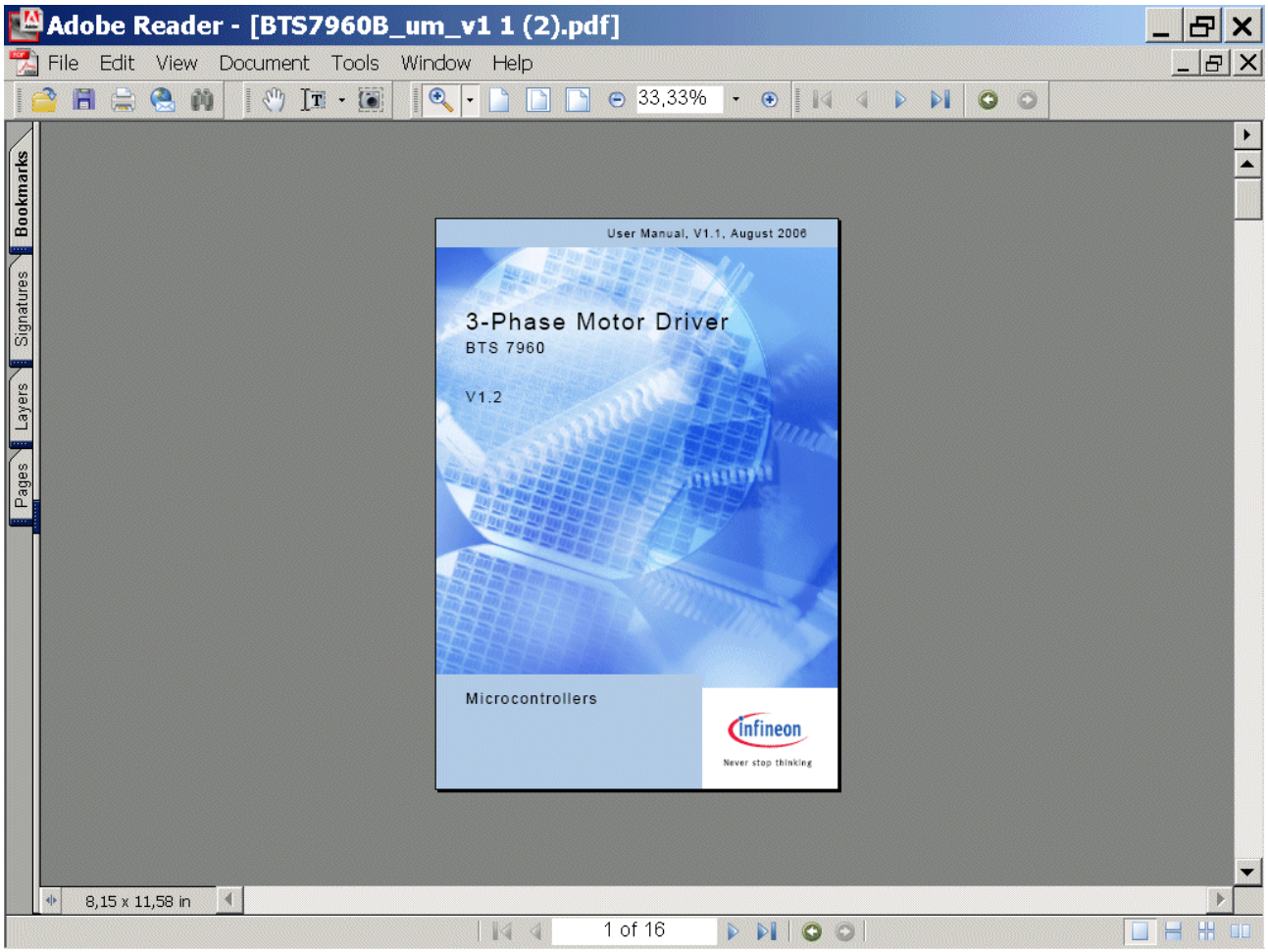
Schematic:


BLDC-Motor
BTS 7960 #3
BTS 7960 #2
BTS 7960 #1
Adobe Reader - [BTS7960B_um_v1 1 (2).pdf]
File Edit View Document Tools Window Help
Options
1 Overview
1.1 Features
1.2 Electrical Characteristics
2 Board Description
2.1 Connectors and Signals
2.1.1 Power Supply Connector X3
2.1.2 Motor Output Connector X4
2.1.3 Hall Interface Connector J1
2.1.4 Easy Kit Connectors
2.2 Mounting Options
2.2.1 Current Sense / Shunt
2.2.2 Hall Sensor Supply
2.2.3 CTRAP Disable
2.2.4 Slew-Rate Adjustment
3 Schematics
4 PCB - Layout
5 Bill of Material
Bookmarks
Signatures
Layers
Pages
Schematics
V1.1, 2006-08-10
User Manual
11 of 16
15 x 11,58 in
50%

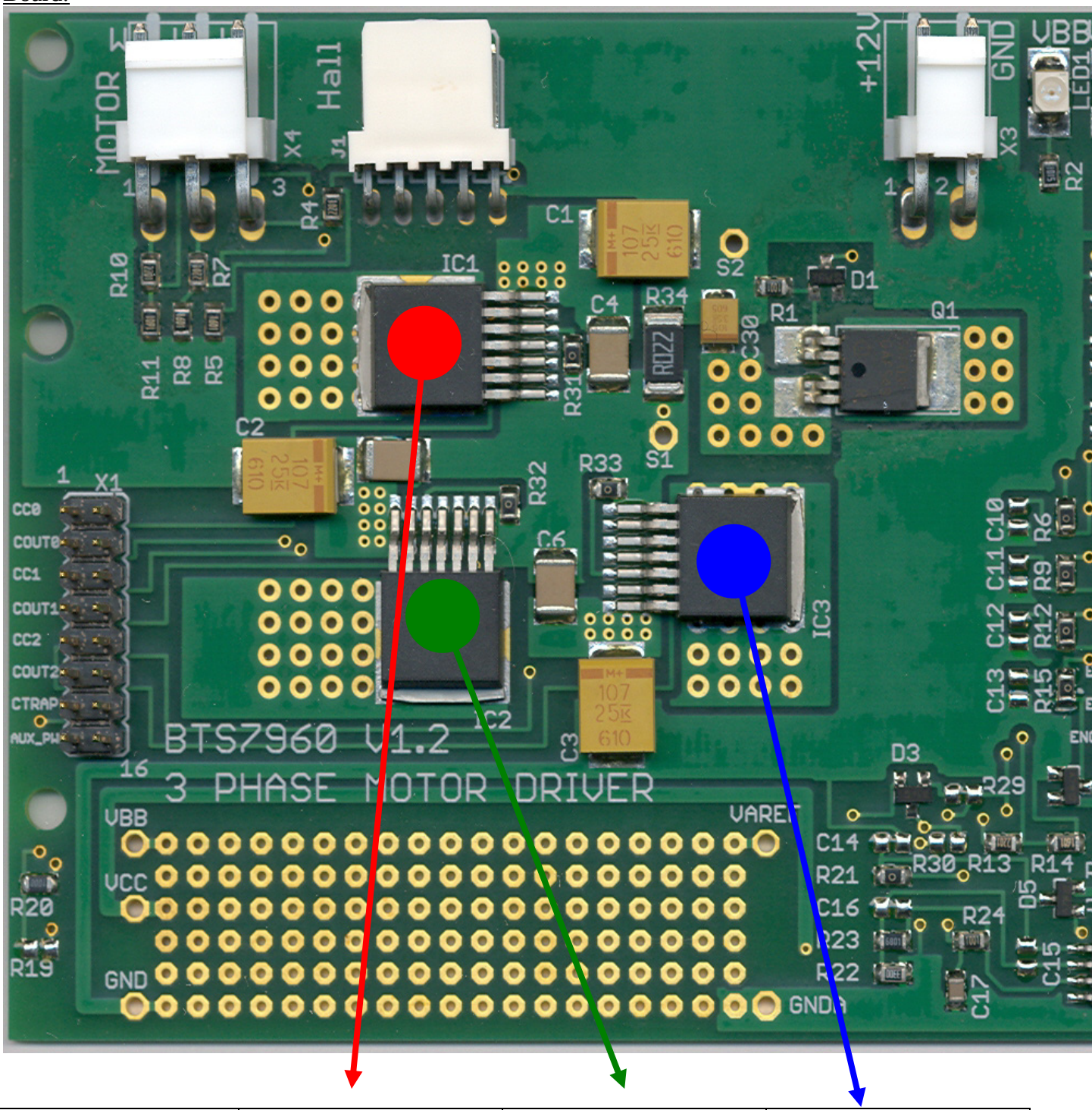


CAPCOM6 Outputs	BTS 7960 #1 (Inputs)	BTS 7960 #2 (Inputs)	BTS 7960 #3 (Inputs)
CC60	IN (high/low-activated)		
COUT60	INH (Inhibit)		
CC61		IN (high/low-activated)	
COUT61		INH (Inhibit)	
CC62			IN (high/low-activated)
COUT62			INH (Inhibit)

Note:
For further information, please refer to the 3-Phase Motor Driver BTS 7960 User Manual.



Board:



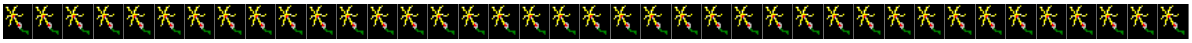
CAPCOM6 Outputs	BTS 7960 #1 (Inputs)	BTS 7960 #2 (Inputs)	BTS 7960 #3 (Inputs)
CC60	IN (high/low-activated)		
COUT60	INH (Inhibit)		
CC61		IN (high/low-activated)	
COUT61		INH (Inhibit)	
CC62			IN (high/low-activated)
COUT62			INH (Inhibit)

12.) Thanks To

Ursula, Arno, Christian, Robert and Ronny for their support.

13.) Feedback (XC164CM+BLDC):

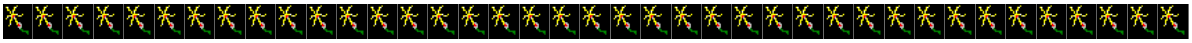
Your opinion, suggestions and/or criticisms:



Contact Details (this section may remain empty should you wish to offer feedback anonymously):

If you have any suggestions please send this sheet back to:

email: mcdocu.comments@infineon.com
FAX: +43 (0) 4242 3020 5783



Your suggestions:

http://www.infineon.com

Published by Infineon Technologies AG