ICC PROTOCOL REFERENCE MANUAL

INTRODUCTION

This document is an advanced technical reference manual. It has been written for engineers who want to build their own ST7 programming, testing or debugging tools using the In-Circuit Communication (ICC) protocol. This document does not target engineers who only use standard development tools such as ST programming boards or debuggers.

This manual describes how to communicate with an ST7 microcontroller using the In-Circuit Communication protocol. For implementation details such as register or stack top address, refer to product datasheets.

The **In-Circuit Communication (ICC)** protocol enables an ST7 microcontroller to communicate with an External Controller (e.g. PC with ICC interface board) with only 4 wires including V_{SS}. This protocol is used to download a program into the RAM for execution. To perform the ICC communication, the ST7 executes an ICC Monitor stored into the write protected ST7 System Memory. Consequently, the ICC protocol performs three different applications described below:

– ICP: In-Circuit Programming is an ST7 FLASH programming method using the ICC communication protocol. ICP is used to update the entire contents of the FLASH memory (including option bytes). For more details, see “FLASH Programming” reference and quick reference manuals.

– ICT: In-Circuit Testing is a flexible method of performing production test routines that can be easily modified and extended without affecting the content of the FLASH program memory (see Section 10).

– ICD: In-Circuit Debugging is the ability to debug a FLASH device using the ICC protocol. This feature allows the implementation of a low cost emulator strategy (see Section 9).

This manual describes the In-Circuit Communication (ICC) protocol and its implementation with an External Controller for performing In-Circuit Programming (ICP) or In-Circuit Testing (ICT) of ST7 FLASH microcontrollers (MCUs) or for performing In-Circuit Debug (ICD) with the ST72F264 for example.

ST72Cxxx devices do not feature ICC protocol. In these devices, the In-Situ Programming (ISP) method with a different protocol is used. ISP is not described in this document. For more information on programming ST72Cxxx devices, refer to the AN1179 application note.

Related Documentation

- Flash Programming Reference Manual
- Flash Programming Quick Reference Manual[1]
- Debug Module Reference Manual

[1]The Quick Reference manual provides device-specific tables included in the previous revision of the Flash Progamming and ICC reference manuals.

Rev. 3

Table of Contents

Table of Contents

GLOSSARY

This chapter gives a brief definition of all new acronyms and names linked to the ICC protocol as a quick reference:

- **XFlash (Extended)**: Extended FLASH memory is based on EEPROM technology. The XFlash provides extended features such as byte-by-byte re-programming (by means of byte erasing) and data EEPROM capability. The XFlash devices are available between 1k and 16kbytes.
- **HDFlash (High Density)**: High density FLASH memory is based on FLASH technology. The high density of the HDFlash cell is used for devices with 4k up to 60kbytes FLASH memory. This HDFlash is programmed byte per byte but erased by sector.
- **IAP (In-Application Programming)**: The IAP is the ability to re-program the FLASH memory of a microcontroller while the device is already plugged-in to the application and the application is running.
- **ICC (In-Circuit Communication)**: The ICC is either an ST7 mode or a software protocol stored by ST7 micrcontrollers in the System Memory. The ICC software protocol is used to download programs into RAM and execute them. It is used for ICP. It is accessed by entering ICC Mode after a dedicated reset sequence.
  - **ICC Monitor**: The ICC Monitor is the ST7 software which manages the ICC protocol. It is located in System Memory.
  - **ICC Mode**: The ICC Mode is a special ST7 run mode which executes the ICC Monitor after a reset sequence. It is entered by means of the $\overline{\text{RESET}}$ pin and with a specific reset sequence. This mode remains active until the next reset sequence.
  - **ICC Session**: An ICC Session takes place when the ST7 ICC Monitor is executed in USER or ICC Mode.
- **ICP (In-Circuit Programming)**: The ICP is the ability to program the FLASH memory of a microcontroller using ICC protocol while the device is already plugged-in to the application.
- **ICT (In-Circuit Testing)**: The ICT is a flexible method of performing production test routines that can be easily modified and extended without affecting the content of the FLASH program memory.
- **ICD (In-Circuit Debugging)**: The ICD is the ability to debug a FLASH device using the ICC protocol. This feature allows low cost emulator strategy.
- **USER Mode**: The USER Mode is the standard user application run mode in the ST7. It is entered by means of the $\overline{\text{RESET}}$ pin and without any specific reset sequence. This mode remains active until the next reset sequence.
- **RESET Phase**: The ST7 RESET phase is the time spent between the rising edge of the $\overline{\text{RESET}}$ pin and the execution of the first instruction
- **Embedded Commands**: The Embedded Commands are the software drivers used by the USER application to program or erase HDFlash devices.
- **System Memory**: The System Memory is a write-protected part of the memory which contains the ICC Monitor. The HDFlash System Memory also contains the Embedded Command software. The XFlash System Memory is accessible when the USER application is executed. However, this is not the case for HDFlash System Memory.

1 OVERVIEW

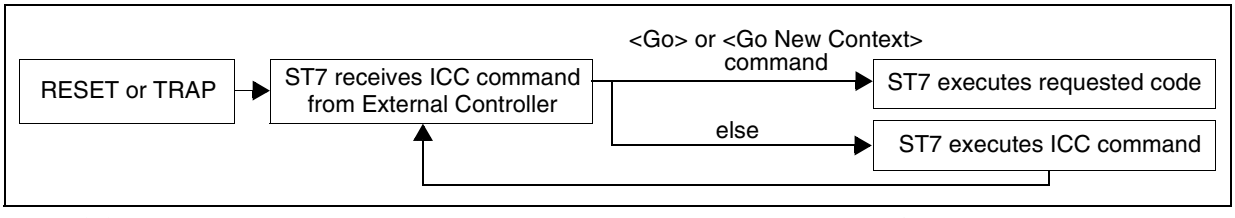
ICC defines a half-duplex synchronous serial communication protocol with an automatic adaptation to the speed of the slower device (see Section 4). The communication is ensured by two lines: ICCCLK and ICCDATA.

Data is exchanged byte by byte. The bytes sent by the External Controller are commands for the ST7 to execute, following an higher-level protocol. The command library allows a wide range of actions (see Section 5).

To initialize the ST7 to be able to manage this protocol, a special ST7 functional mode has to be entered. This mode is called “ICC Mode”. When this mode is selected, the ST7 executes an ICC Monitor program which allows the ST7 to receive commands to be executed using the ICC protocol (implemented as a software communication peripheral). The flowchart shown in Figure 1 gives an overview of the ICC management.

Note: With XFlash and ROM ST7 devices, the ICC Monitor can also be executed in “USER Mode”. See section 6 on page 26 for more details.

Figure 1. ICC Protocol Flowchart

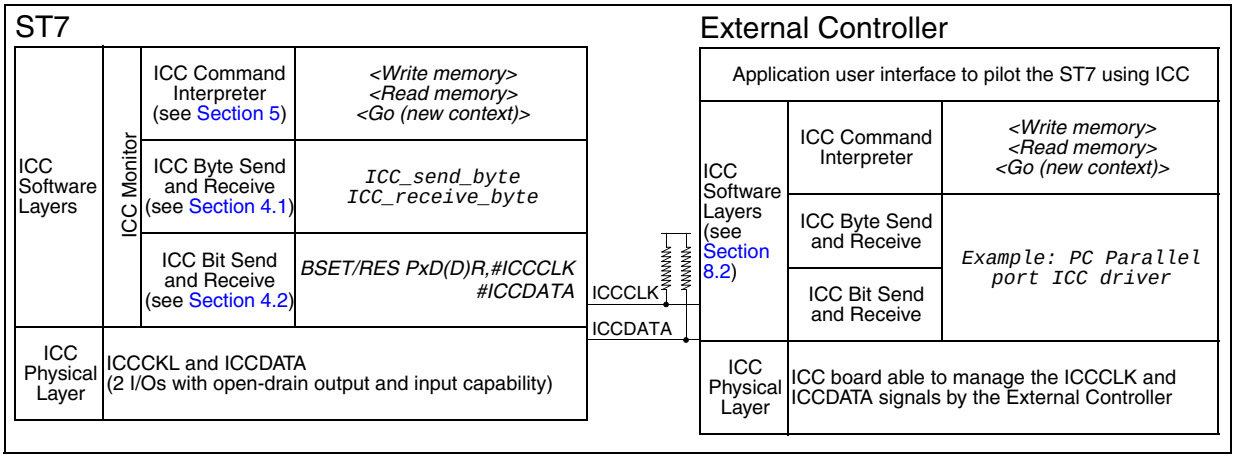


The ICC protocol can be divided into four different layers on the ST7 side and five on the External Controller side, as shown in Figure 2.

Caution: During ICC Mode, the I/O, TLI and peripheral interrupt vectors are not available. Consequently, they cannot be used. Only the HDFlash embedded command non-maskable interrupt is available.

For an overview of the Application Board hardware description, refer to the device datasheet or Figure 19 "PC to ICC Interface Board - ICCCLK and ICCDATA Signals" on page 35.

Figure 2. ICC Protocol Layers



1.1 IMPLEMENTED ICC FEATURES

Refer to the FLASH Programming Quick Reference Manual for a table of implemented ICC features and their main characteristics for each ST7 device. The data mentioned in this table is described in detail later on in this document.

2 COMMUNICATION TIMINGS

Description		Conditions	Min.	Max.	Unit
ICC receive	Baud rate [1]	f_{CPU}=8MHz [3]		181 / 22.6	kbps / kBps
	1KByte transfer time [2]		50		ms
ICC transmit	Baud rate [1]			133 / 16.6	kbps / kBps
	1KByte transfer time [2]		69		ms

Commands are simple in order to be fast. For example, in the same conditions as above, the External Controller can read or write directly into the ST7 RAM or registers at around 15 kBps.

Notes:

1. This maximum baud rate is the fastest communication speed at bit transmission level. It is given in bits per second (bps) and bytes per second (Bps).
2. This transfer time takes into account the added transferred bytes for the ICC command management.
3. The speeds and timings given in this table are proportional to ST7 f_{CPU} frequency and assume that the External Controller is faster than the ST7. f_{CPU} = 8MHz is given as an example.

3 ENTERING ICC MODE

Two different sequences exist for making the ST7 enter ICC Mode depending on the selected device. Both methods are based on a pulse counter in the ST7 which determine whether the microcontroller enters ICC Mode or not. Refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual to know which method has to be applied to the ST7 devices.

3.1 CONTROLLED WINDOW ICC MODE ENTRY METHOD

To make the ST7 enter ICC Mode, three steps have to be performed:

- First the pulse counter must be reset: to do this, $\overline{\text{RESET}}$ must be kept low while ICCCLK is kept high or ICCSEL is kept low depending on the device (refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual for more details).
- Then keeping $\overline{\text{RESET}}$ low, N pulses must be performed on ICCDATA while ICCCLK is kept low or ICCSEL is kept high depending on the device. At the end of the pulse sequence, ICCCLK and ICCDATA must be kept low (refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual) .
- Finally $\overline{\text{RESET}}$ is released (high state) to start the ICC Monitor execution after the ST7 internal reset phase (256 or 4096 t_{CPU} periods).

Special case: Devices without ICCSEL pin

When ICCCLK is used to reset the pulse counter (and not ICCSEL), the ST7 ICCCLK pin is configured as a pull-up input as long as $\overline{\text{RESET}}$ is in low state. This guarantees that the ST7 enters USER Mode when the ICCCLK pin is unconnected or only connected to high impedance outputs.

Caution: In the above case, the ICCCLK pin must always be pulled up. This is to avoid entering ICC mode unexpectedly during normal application operation.

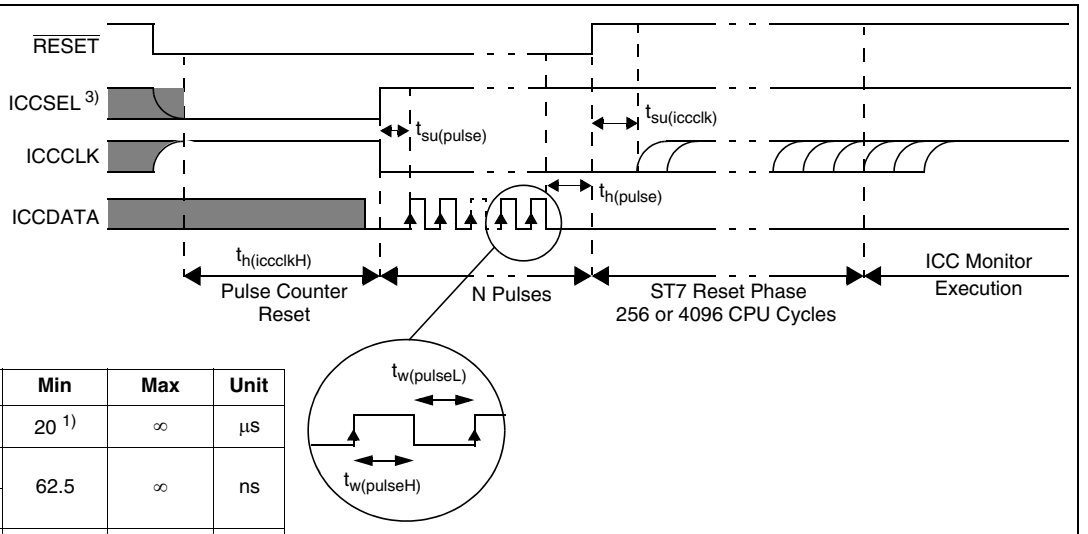
In this case, the ST7 ICCSEL pin is always configured as a floating input.

The ST7 ICCDATA pin is configured as a standard I/O (floating input) as long as $\overline{\text{RESET}}$ is in low state and during the ST7 reset phase.

The pulse sequence is totally independent from the ST7 CPU clock (fully asynchronous ICC Mode entry). The selected ST7 mode (USER or ICC) depends on the pulse counter value when $\overline{\text{RESET}}$ returns to high state.

Figure 3 shows a typical sequence for entering ICC Mode.

Figure 3. Entering ICC Mode with Controlled Window



Symbol	Min	Max	Unit
$t_{h(iccclkH)}$	20 1)	∞	µs
$t_{w(pulseL)}$	62.5	∞	ns
$t_{w(pulseH)}$			
$t_{su(pulse)}$	0.5	∞	t_{CPU}
$t_{h(pulse)}$			
$t_{su(iccclk)}$	1		µs

Notes:

1. This hold time, required to reset the pulse counter, is directly linked to the $\overline{\text{RESET}}$ pad filter characterized by $t_{h(RSTL)in}$ (see device datasheet for more details).
2. A glitch on ICCSEL or ICCCLK (depending on the device, refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual) while $\overline{\text{RESET}}$ is pulled low could reset the pulse counter. A glitch on ICCDATA pad could be interpreted as a pulse during the “N Pulses” phase. Therefore, noise must be avoided on these pins.
3. On dual voltage HDFlash devices, ICCSEL and V_{PP} (FLASH programming voltage) are on the same pin.

Caution: After the RESET phase, ICCSEL(/V_{PP}) pin must be kept high (V_{DD} or 12V for HDFlash) to avoid ICC communication issues.

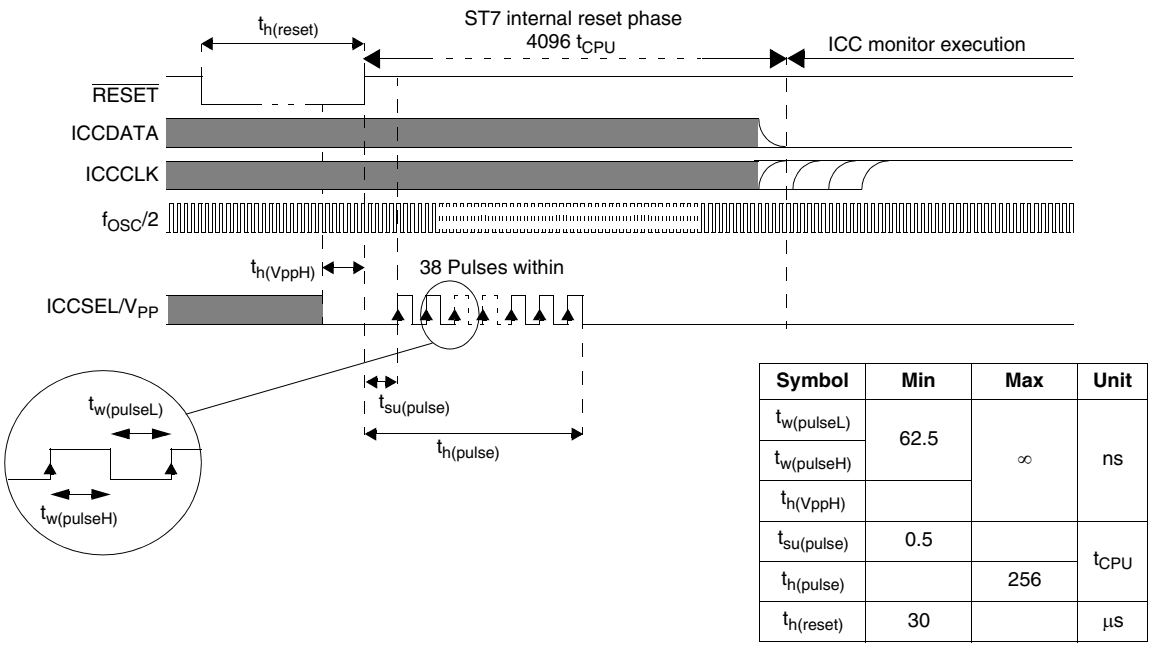
3.2 FIXED 256 T_{CPU} WINDOW ICC MODE ENTRY METHOD

A pulse sequence is sent by an External Controller to the ICCSEL/V_{PP} ST7 pin within a time window of 256 t_{CPU} after releasing the $\overline{\text{RESET}}$ pin. To make the ST7 enter ICC Mode, the External Controller must generate 38 pulses on the ICCSEL/V_{PP} line. The ST7 pulse counter is reset when the $\overline{\text{RESET}}$ pin is pulled low. The pulse sequence is dependent from the ST7 CPU clock.

Note: In devices with no $\overline{\text{RESET}}$ pin (such as the ST7SCR), the start of the pulse sequence must be synchronized with V_{DD}. Refer to the Flash Programming Quick Reference Manual for device-related data.

Figure 4 shows a typical sequence for entering ICC Mode.

Figure 4. Entering ICC Mode with Fixed t_{CPU} Window



Symbol	Min	Max	Unit
$t_{w(pulseL)}$	62.5	∞	ns
$t_{w(pulseH)}$			
$t_{h(VppH)}$			
$t_{su(pulse)}$	0.5		t_{CPU}
$t_{h(pulse)}$		256	
$t_{h(reset)}$	30		µs

3.3 EXTERNAL CLOCK/APPLICATION CLOCK ICC MODES

As shown in “Implemented ICC Features” table in the FLASH Programming Quick Reference Manual, on some devices, depending on the number of pulses counted for ICC mode entry, ICC mode is either in External Clock or Application Clock mode.

- In External clock mode, the user option byte settings are ignored,
- In Application clock mode, the user option byte settings are taken into account.

In External clock mode, reserved Flash locations (which may contain internal oscillator calibration data for example) are not write protected, and could be corrupted if you program the device using a wrong address. In Application clock mode, reserved Flash locations are write protected, so it is safer than External clock mode.

3.3.1 Special Case of Hardware Watchdog Option

If the hardware watchdog option has been enabled in the device to be reprogrammed, a reset occurs after the watchdog timeout, preventing the successful completion of programming. This issue has been solved for most devices, a modification to their design causes them to automatically switch the hardware watchdog option to software watchdog in ICC mode. The only device exceptions to this are shown in the “Implemented ICC Features” table in the FLASH Programming Quick Reference Manual. For these devices, the ST7 has to enter external clock mode in order to be able to reprogram devices with hardware watchdog enabled. To select external clock mode, send 35 pulses instead of the 38 as shown in the same table. In the case of the ST72F651, the effect of sending 35 pulses will cause the watchdog setting to be ignored, but other user options are taken into account, so it is not identical to external clock mode as defined in Section 3.3.

4 ICC BYTE EXCHANGE PROTOCOL

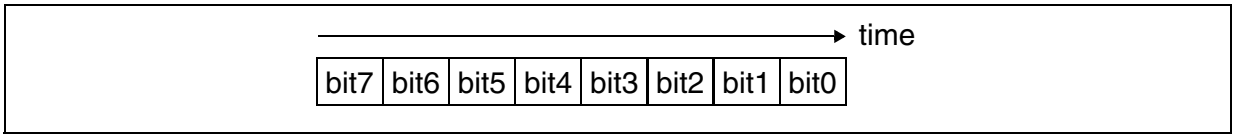
The ICC protocol defines a synchronous serial communication where both interlocutors are partial masters of the clock. The purpose of such a protocol is to allow communication even if the clock frequencies of both devices are different. ICCCLK carries the clock signal and ICCDATA the data signal: the logic level on ICCDATA is valid only when ICCCLK is low.

This communication is based on an open-drain line approach. Both lines are always connected to a pull-up resistor located inside the External Controller. For the Controller, “releasing” ICCCLK or ICCDATA means connecting the line only to this resistor.

4.1 BYTE EXCHANGE FLOW

Data is transmitted byte by byte, with the most significant bit being sent first as shown in Figure 5.

Figure 5. ICC Byte Definition



This means that the transmission direction is the same for 8 consecutive bits. So, the emitting device does not need to release ICCDATA between two bit transmissions. Actually, in the ICC Monitor, the ST7 releases ICCDATA only after sending the last bit of the byte, i.e. the LSB.

In the following Figure 6 and Figure 7, ICCDATA is always released between two bit transmissions, for figure clarity.

4.2 BIT EXCHANGE FLOW

The following describes how a bit is transmitted from one device to another during an ICC Session.

During the interval between two bit exchanges, both lines are released by both devices, so they are both in high state. The exchange is always initiated by the ST7 when it is ready to receive or send the bit, and when it has detected that ICCCLK is high. The signal sequence depends on the transmission direction.

Transmission direction and the end of ICC Session are determined by the ICC command protocol level, described in Section 5.

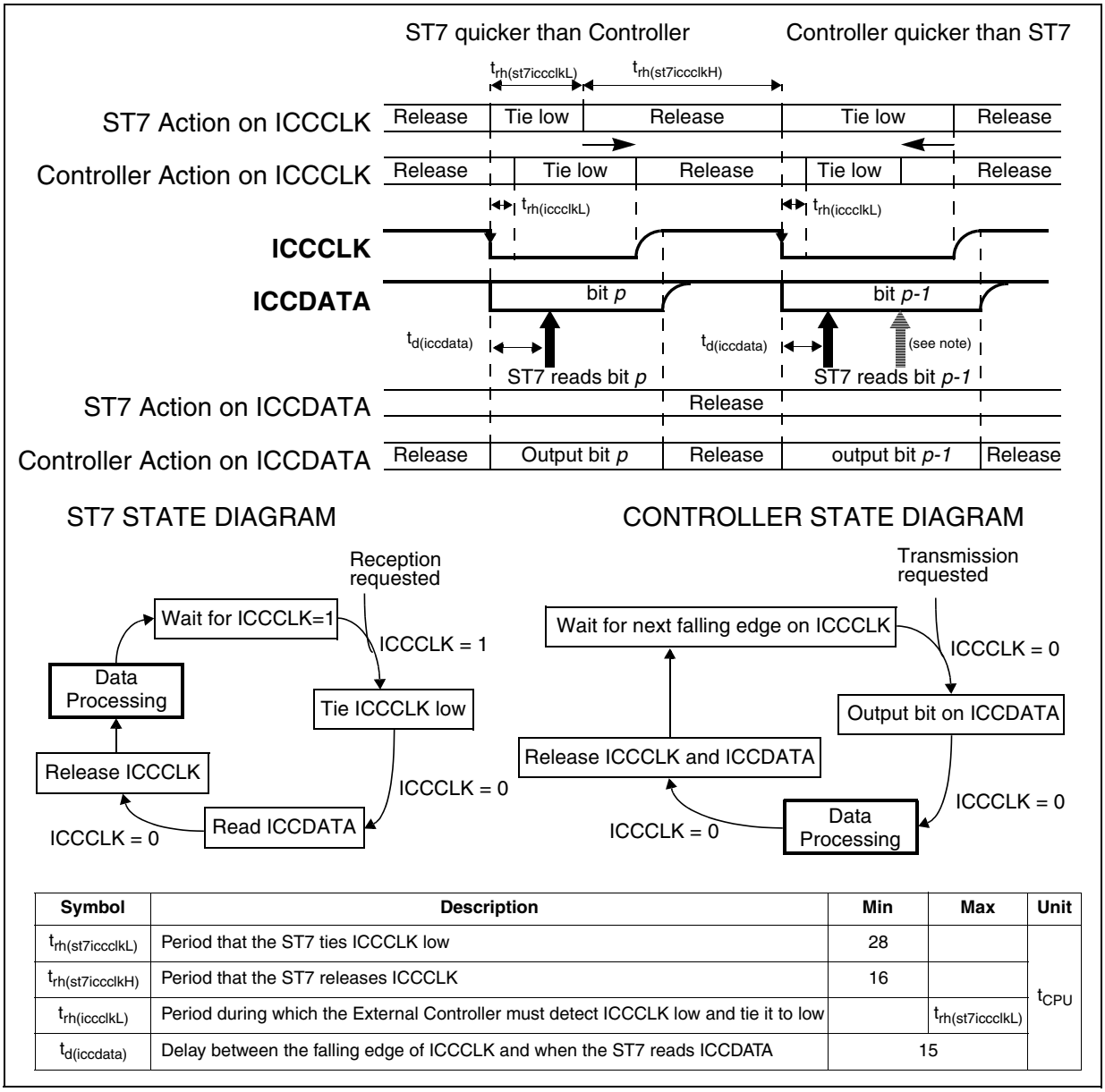
To understand in detail how the ST7 follows this physical protocol, refer to the ICC Monitor code in Section 6. Notice that the External Controller must respect certain time constraints imposed by the ST7. Section 8 describes an example of hardware implementation that meets these constraints.

Note: In each of the following state diagrams, the data processing state corresponds to the transition between two bit exchanges. The protocol works even if this state lasts for a long time. When the ST7 is in this state, ICCCLK can be low or high. When the Controller is in this state, ICCCLK is always low.

4.2.1 Next Bit is to be RECEIVED by the ST7 Device

If the next bit is to be received by the ST7, the MCU starts by tying the clock low. See Figure 6. In reaction to this falling edge, the External Controller outputs the data bit on ICCDATA and ties the clock low. After a $t_{d(iccdata)}$ delay, the ST7 reads the data bit on ICCDATA. Then it releases the clock ($t_{rh(st7iccclkL)}$ after the ICCCLK falling edge). When the External Controller is ready to receive or send the next bit, it releases the clock. When ICCCLK returns to high state, the External Controller releases ICCDATA (see Section 4.1 Byte Exchange Flow). Then the next bit exchange is possible.

Figure 6. ICC Bit Exchange when Bit is received by the ST7



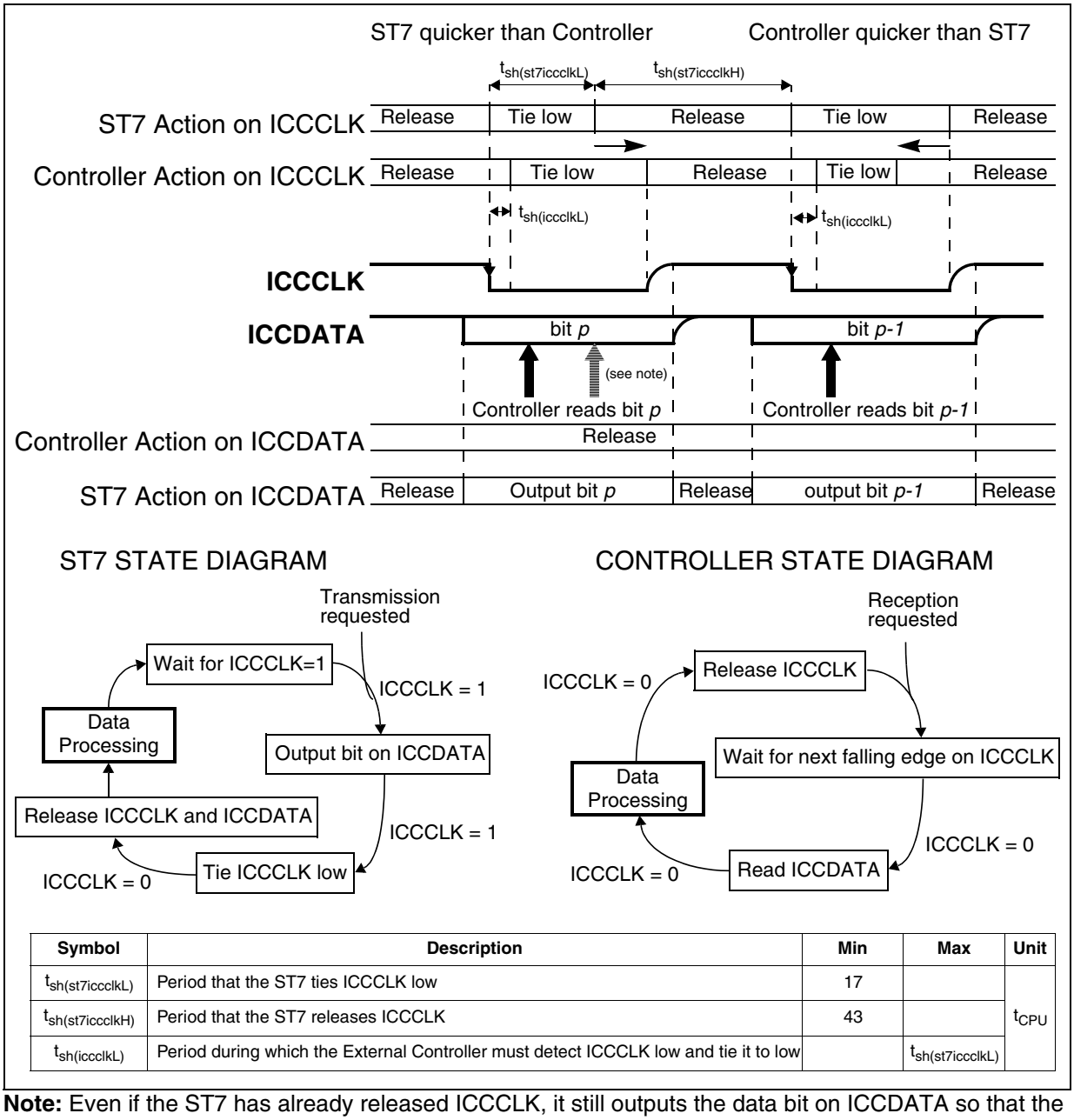
Symbol	Description	Min	Max	Unit
$t_{rh(st7iccclkL)}$	Period that the ST7 ties ICCCLK low	28		t_{CPU}
$t_{rh(st7iccclkH)}$	Period that the ST7 releases ICCCLK	16		
$t_{rh(icccllkL)}$	Period during which the External Controller must detect ICCCLK low and tie it to low		$t_{rh(st7iccclkL)}$	
$t_{d(iccdata)}$	Delay between the falling edge of ICCCLK and when the ST7 reads ICCDATA	15		

Note: Even if the Controller has already released ICCCLK, it still outputs the data bit on ICCDATA so that the ST7 can read it.

4.2.2 Next Bit to be SENT by the ST7 Device

If the next bit is to be sent by the ST7, the MCU outputs the data bit on ICCDATA. See Figure 7. Then it ties the clock low. In reaction to this falling edge, the External Controller ties the clock low and reads the data bit on ICCDATA. After $t_{sh(st7iccclkL)}$, the ST7 releases the clock. When the External Controller is ready to receive or send the next bit, it releases the clock. When ICCCLK returns to high state, the ST7 releases ICCDATA (see Section 4.1 Byte Exchange Flow). Then the next bit exchange is possible.

Figure 7. ICC Bit Exchange when Bit is sent by the ST7



Symbol	Description	Min	Max	Unit
$t_{sh(st7iccclkL)}$	Period that the ST7 ties ICCCLK low	17		t_{CPU}
$t_{sh(st7iccclkH)}$	Period that the ST7 releases ICCCLK	43		
$t_{sh(icccclkL)}$	Period during which the External Controller must detect ICCCLK low and tie it to low		$t_{sh(st7iccclkL)}$	

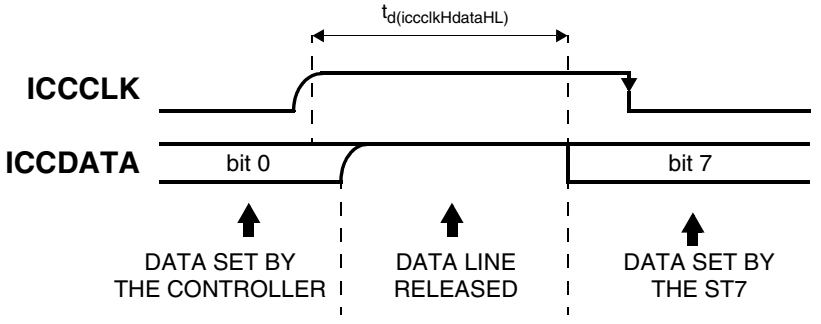
Note: Even if the ST7 has already released ICCCLK, it still outputs the data bit on ICCDATA so that the Controller can read it.

4.2.3 Switching from ICC Receive to ICC Send

The Figure 8 describes how ICCCLK and ICCDATA are managed when switching from a bit received by the ST7 to a bit to be sent by the ST7.

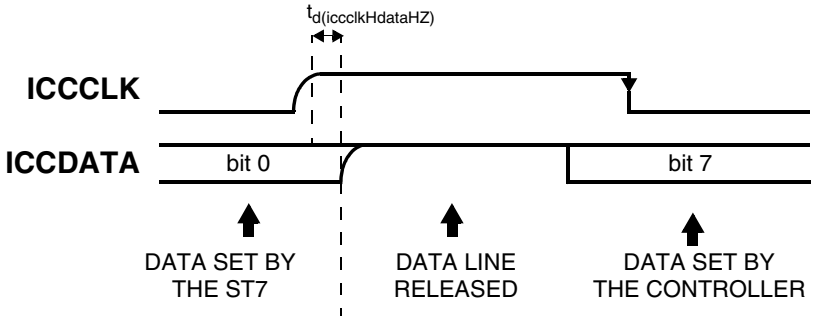
The Figure 9 describes how ICCCLK and ICCDATA are managed when switching from a bit sent by the ST7 to a bit to be received by the ST7.

Figure 8. Switching from ICC Receive to ICC Send



Symbol	Description	Value	Unit
$t_{d(iccclkHdataHL)}$	Delay between the ST7 detects ICCCLK high and when the ST7 sets the data on ICCDATA	42	t_{CPU}

Figure 9. Switching from ICC Send to ICC Receive



Symbol	Description	Value	Unit
$t_{d(iccclkHdataHZ)}$	Delay between the ST7 detects ICCCLK high and when the ST7 releases ICCDATA	11	t_{CPU}

4.3 BEGINNING OF AN ICC SESSION

The ST7 RESET phase is the time spent between the rising edge of the $\overline{\text{RESET}}$ pin and the execution of the first instruction (see Section 3 for more details).

When the ST7 has finished its RESET phase, it jumps into the ICC Monitor if ICCDATA is forced low by the Controller. So the External Controller must tie low ICCDATA while the ST7 is still in RESET phase.

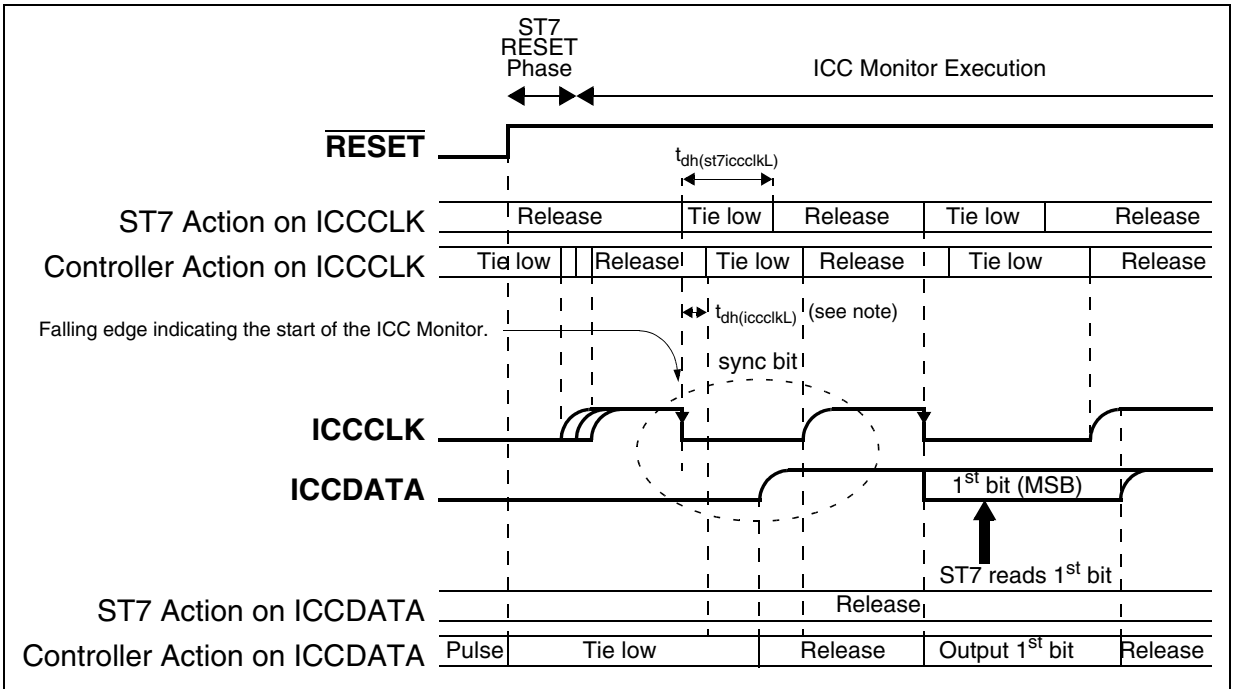
Then the ST7 waits for a high level on ICCCLK. So the External Controller may release ICCCLK while the ST7 is still in RESET phase or afterwards. The protocol works in both cases.

To make sure both devices are synchronized, the first bit exchanged is a dummy, i.e. during a single clock period (a falling and a rising edge on ICCCLK), the ICCDATA value is ignored. So the External Controller can release ICCDATA line at any time while it ties ICCCLK low.

After this synchronization bit, the ST7 receives the first valid data bit from the Controller which corresponds to the MSB of the first ICC command (see Section 5).

The beginning of the ICC Session following an ICC Mode entry is shown in Figure 6. Other methods of starting an ICC Session exist, see Section 9 for more details.

Figure 10. Beginning of ICC Session when the ICC Monitor is called by ICC Mode Entry



Symbol	Description	Min	Max	Unit
$t_{dh(st7iccclkL)}$	Period that the ST7 ties ICCCLK low	16		t_{CPU}
$t_{dh(iccclkL)}$	Period during which the External Controller must detect ICCCLK low and tie it to low		$t_{dh(st7iccclkL)}$	

Note: The External Controller must tie the ICCCLK line low before a delay of $t_{dh(iccclkL)}$. This delay corresponds to the minimum time needed by the ST7 to release the ICCCLK line.

5 ICC COMMAND LIBRARY

5.1 IMPLEMENTATION

The ICC is a command-based protocol. After a reset, following the sync bit, the ICC Monitor waits for a byte from the External Controller. This byte represents the code of a command. The ST7 executes this command. This may imply other byte exchanges, in both directions. After the ST7 has executed the command, it waits for another one. If the command code does not correspond to any command, the ST7 ignores it and waits for the next one (It reads all the following bytes until it finds a valid command code).

Transition between two ICC commands: After sending or receiving a byte, the ST7 continues to execute the ICC Monitor when ICCCLK is released after the least significant bit. Therefore at the end of an ICC command, as the External Controller must keep the ICCCLK line low until it has prepared the first bit of the next command to be sent, the current ICC command can remain unexecuted by the ST7. To solve this issue, a dummy ICC command (FFh code for example) could be sent after the ICC command to be executed (see “ST7 action” description in section 5.5 on page 23 for more details).

ST has defined 3 variants of ICC Monitors, with a different set of commands as shown in Table 1: The “Basic” variant is used in small XFlash devices without a debug module and HDFlash devices to optimize the System Memory size; the “Medium” variant is used in ROM devices to support easy memory dump; the “Advanced” variant is used only in XFlash devices which embed a debug module. To know which variant is used in a device, refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual .

Table 1. ICC Monitor Commands

	Name	Byte Coding	ICC Monitor Variant		
			Basic (minimum for ICP)	Medium (optimum for ICP)	Advanced (required for ICD)
ICC Commands 1)	Write memory	0xxxxxxxb	X	X	X
	Go	10xxxxxxb	X	X	X 3)
	Go new context				X
	Read memory	110xxxxxb		X	X
Automatic context saving 2)					X

Notes:

1. If both the ST7 and the External Controller try to read a byte at the same time, they both read FFh (because ICCDATA stays high; see Section 4.2 Bit Exchange Flow for details). This behaviour is used to handle the application reset during ICD management (see section 9.1.2 on page 42 for more details).
2. “Automatic Context Saving” is a method to reduce as much as possible the RAM, stack and CPU resources used by the ICC Monitor. For more details, refer to Section 5.4 Automatic Context Saving.
3. With the “Advanced” variant, the first “10xx xxxx” command after ST7 RESET is a <Go> command. Then all other ones are <Go new context> commands.

5.2 ST7 PRODUCT IDENTIFIER

The ST7 product type can be automatically recognized by the External Controller by means of a two byte product identifier. This identifier is stored by the ICC Monitor at the address 0080h/0081h when the first <Go> command after RESET is executed. The first 3 nibbles of this identifier give the ST7 product while the fourth one gives the System Memory version for this product.

– ST7 Product identifier0080h [7:0], 0081h [7:4]

– System Memory version0081h [3:0]

5.3 RAM AND CPU REGISTERS USED BY AN ICC SESSION

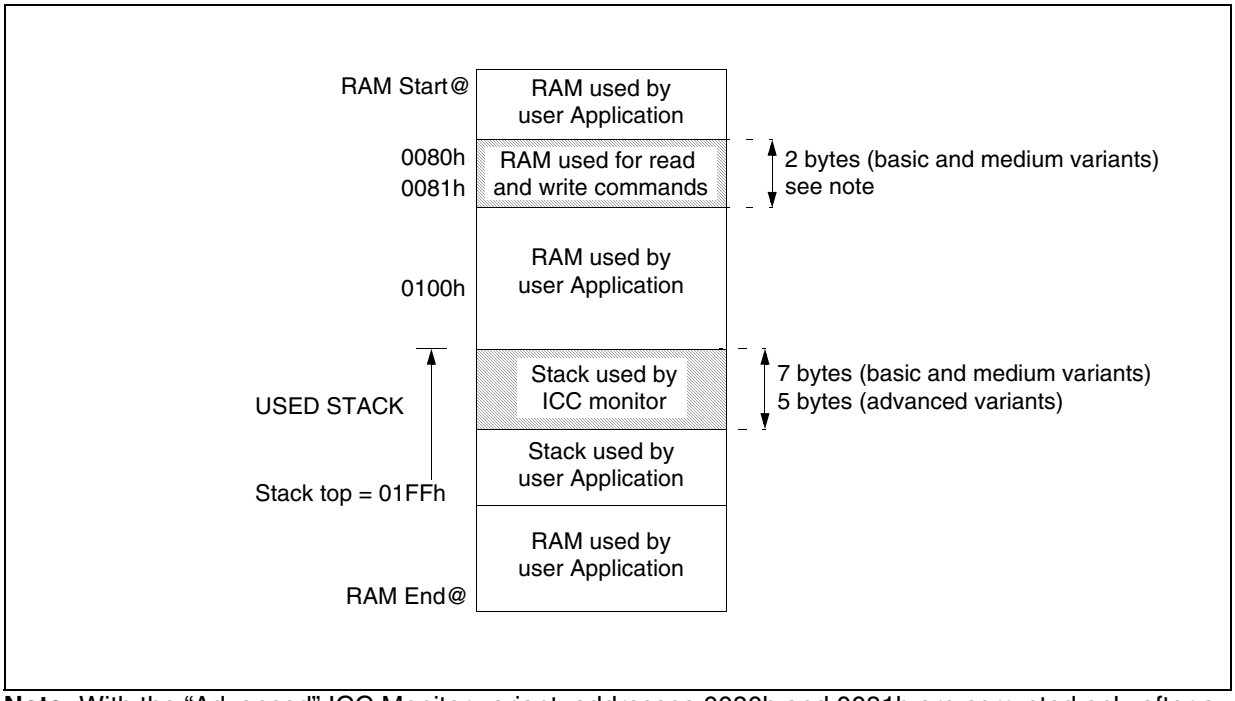
To execute the ICC Monitor, the ST7 needs to use standard RAM and stack locations, and CPU register resources as shown in Figure 12 and explained in this section. To reduce this impact, the advanced ICC variant includes the “Automatic Context Saving” feature described in Section 5.4 Automatic Context Saving.

The ST7 enters the ICC Monitor just like a TRAP interrupt routine: PC, X, A and CC registers are stacked. The <Go> and <Go new context> commands perform an IRET (Interrupt Return) instruction. With the <Write memory> command, the External Controller can directly modify the contents of the stack. This way, when a <Go> or <Go new context> command is executed, the ST7 core registers can be loaded with any given value. A useful application of this method is entering a certain value in the Program Counter, which makes the ST7 jump to a given program location in ROM or in RAM.

Executing a command uses one layer of subroutine calls, i.e. two stack locations. This means that the ICC Monitor could occupy up to seven bytes in the stack (see Figure 12). In ICD, stack occupation is critical. Consequently, the ICC Monitor starts the ICC Session with an automatic context saving: the ST7 unstacks the two last registers (A and CC) and sends them to the Controller. At the end of the session, the <Go new context> command retrieves the registers from the Controller and restacks them before performing the IRET. This way, only five bytes are necessary.

The <Write memory> and <Read memory> commands modify the Y register. They also overwrite two zeropage RAM addresses: 0080h and 0081h. The automatic context saving sends the values of these three resources to the External Controller. The <Go new context> command retrieves these values from the Controller before performing the IRET.

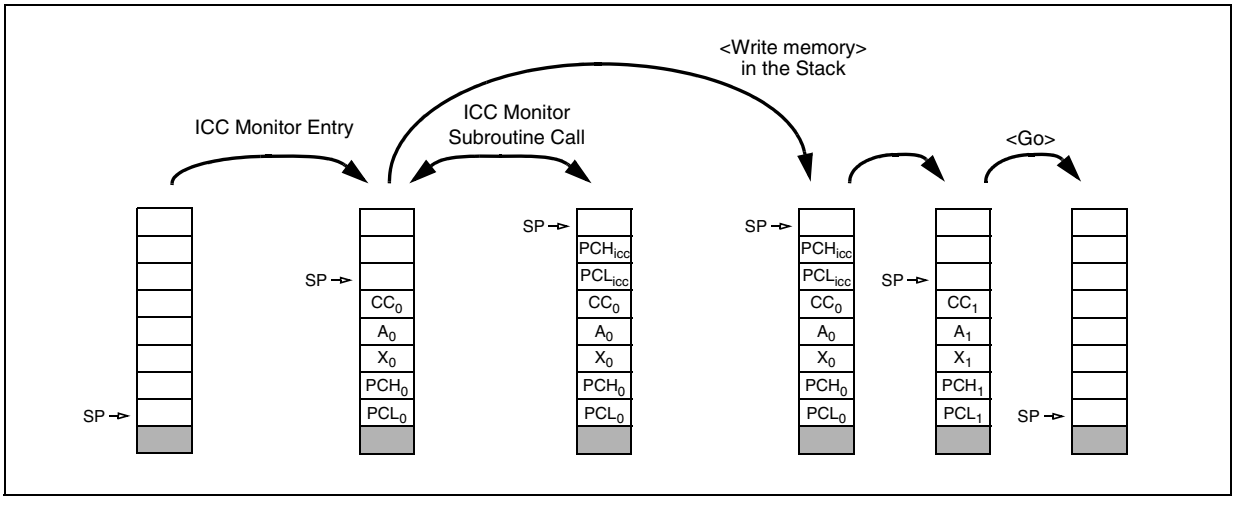
Figure 11. RAM used by an ICC Monitor Execution



Note: With the “Advanced” ICC Monitor variant, addresses 0080h and 0081h are corrupted only after a RESET phase and a <Go> command. Then, after a <Go new context> command these two RAM location are no longer corrupted by the ICC Monitor.

Unless otherwise specified, an ICC command modifies all the ST7 core registers except Y but they are restored when exiting the ICC Monitor with a <Go> command.

Figure 12. Stack Manipulation without Automatic Context Saving



5.4 AUTOMATIC CONTEXT SAVING

When using ICC for In-Circuit Debugging (ICD), as few resources as possible have “to be stolen” from the user application in order to be as close as possible to the final software application.

“Automatic Context Saving” is a method which has been implemented in the advanced ICC variant in order to support ICD management.

When automatic context saving is available, after an ICD abort or a breakpoint, the ST7 starts the ICC Session by sending a 6-byte header (see definition below). Then it waits for the first command code. The <Go new context> command allows the ST7 to exit this ICC Session and receive the 6 bytes previously sent.

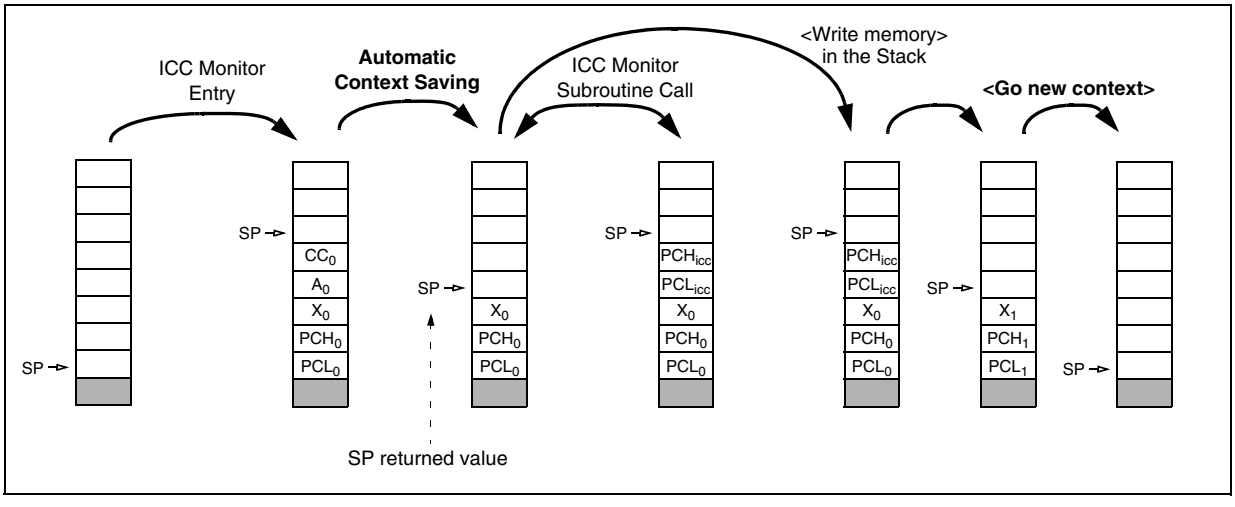
Following a reset sequence, the ICC Session does not manage automatic context saving (i.e. the <Go> command is available instead of the <Go new context> command).

Definition: The ST7 sends the contents of its Y register to the External Controller. Then it retrieves the two last register bytes (CC and A) from the stack that were saved when the ST7 entered ICC Monitor (see Figure 13), and sends them to the Controller. Finally, it sends the contents of the 0080h and 0081h zeropage RAM addresses and the current stack pointer value (S). At the end of the ICC Session, the context is restored receiving the 6 bytes previously sent (see Section 5.5.3 <Go New Context> Command for more details).

Direction	ST7 → Controller					
Data	Y	CC	A	Data (0081h)	Data (0080h)	S
ST7 Action						

Note: Automatic context saving modifies the Y register after sending its previous value to the Controller.

Figure 13. Stack Manipulation with Automatic Context Saving



5.5 COMMAND DESCRIPTIONS

5.5.1 <Write memory> Command

The External Controller writes up to 129 adjacent bytes anywhere in the ST7 addressing space.

Direction	Controller → ST7							
Data	Command Code [0xxxxxxxb]	@ MSByte	@ LSByte	N-1	Data (@+N-1)	Data (@+N-2)	...	Data (@)
ST7 Action						➥ Data write in memory at address @+N-1	➥ Data write in memory at address @+N-2	➥ Data write in memory at address @

The address transmitted after the command code is the lowest address written. The data bytes are written starting at the highest address. In the table above, N is the number of data bytes written. It must not exceed 129 (N-1=80h).

Resource impact (other than the command function):

– Temporarily needs two stack locations.

– Modifies the Y register.

– Overwrites the 0080h and 0081h zeropage RAM addresses. So, make sure that these two addresses are not written to.

Caution: To make the ST7 write the last data byte at address "@" of the device, the Controller must release ICCCLK. To do this, the next ICC command or a dummy command (FFh for example) may be sent by the External Controller.

5.5.2 <Go> Command

The ST7 performs an IRET instruction. This makes it quit the ICC Monitor and execute code at an address which can be located anywhere in memory (i.e. in RAM or in user program memory). This address where the code is executed has to be previously stored in the stack (where the program counter is located) using a <Write memory> command. The CC, A and X registers can be also updated by writing the new value at the corresponding addresses in the stack. After the execution of this command, the ST7 remains in the mode selected by the reset sequence (USER or ICC).

Direction	Controller → ST7
Data	Command Code [10xxxxxxb]
ST7 Action	➥ Perform IRET instruction

Resource impact (other than the command function):

– Temporarily needs two stack locations in addition to the five locations unstacked.

Caution: To make the ST7 execute the IRET instruction, the Controller must release ICCCLK.

5.5.3 <Go New Context> Command

The Controller restores the context previously stored during the automatic context saving. To do this, it fills in the 0080h and 0081h zeropage RAM addresses. Then it fills the S stack pointer (this update allows the Controller application to modify the stack pointer value). Following this stack initialization, the Controller stores two bytes in the stack in order to fill in the A and CC registers at the next interrupt return. Thirdly, it fills in the Y register. Finally, the ST7 performs an IRET instruction. This makes it quit the ICC Monitor and execute code at an address which can be located anywhere in memory (i.e. in RAM or in program memory). This address where the code is executed has to be previously stored into the stack (where the program counter is located) using the <Write memory> command. The X register can be also updated by writing the new value at the corresponding address in the stack. After the execution of this command the ST7 remains in the mode selected by the reset sequence (USER or ICC).

Direction	Controller → ST7							
Data	Command Code [10xxxxxxb]	Data (0081h)	Data (0080h)	S	A	CC	Y	
ST7 Action			➥ Data write in memory at address 0081h	➥ Data write in memory at address 0080h	➥ Stack pointer updated	➥ A & CC pushed into the stack	➥ Y updated and IRET instruction is performed	

Resource impact (other than the command function):

– S stack pointer can be modified.

– Two stack bytes included in the five bytes stacked by the ICC Monitor entry.

Caution: To make the ST7 execute the IRET instruction, the Controller must release ICCCLK.

5.5.4 <Read memory> Command

The External Controller reads up to 129 adjacent bytes in the ST7, regardless of its addressing space.

Direction	Controller → ST7				ST7 → Controller			
Data	Command Code [110xxxxxb]	@ MSByte	@ LSByte	N-1	Data (@+N-1)	Data (@+N-2)	...	Data (@)
ST7 Action					➥ Data read in memory at address @+N-1	➥ Data read in memory at address @+N-2		➥ Data read in memory at address @

The address transmitted after the command code is the lowest address read. The data bytes are read starting at the highest address. In the table above, N is the number of data bytes read. It can not exceed 129 (N-1=80h).

Resource impact (other than the command function):

– Temporarily needs two stack locations.

– Modifies the Y register.

– Overwrites the 0080h and 0081h zeropage RAM addresses.

5.5.5 ICC Command Summary

<Write memory> → [0xxxxxxxb]	@ MSByte →	@ LSByte →	N-1 →	Data(@+N-1) →	...	Data(@) →
<Go> → [10xxxxxxb]						
<Go New Context> → [10xxxxxxb]	Data(0081h) →	Data(0080h) →	S →	A →	CC →	Y →
<Read memory> → [110xxxxxb]	@ MSByte →	@ LSByte →	N-1 →	Data(@+N-1) ←	...	Data(@) ←

Automatic context saving	Y ←	CC ←	A ←	Data(0081h) ←	Data(0080h) ←	S ←

Legend:

→ Data sent by the External Controller to the ST7

← Data sent by the ST7 to the External Controller

6 ICC MONITOR CODE

When the ST7 enters ICC Mode, the reset vector points to the `ICC_monitor_reset` routine. For more details on ICD, refer to section 9 on page 39.

With XFlash and ROM products, the System Memory is accessible either in USER or ICC Modes. Its address is given in a device-specific table in the Flash Programming Quick Reference Manual. Please also refer to section 5.2 on page 20.

The assembler source code of the ICC Monitor is described in APPENDIX 1 on page 51.

7 USING THE ST7 ICC MONITOR

On all ST7 microcontrollers featuring ICC, an ICC Monitor is included in a write protected memory called System Memory. On XFlash devices (e.g. ST72F264) as well as ROM devices, the System Memory is part of the ST7 standard addressing space. On HDFlash devices (e.g. ST72F521), the basic ICC Monitor is embedded in a System Memory located on a parallel addressing space. Consequently it is not accessible while the user application is running.

The ST7 executes this monitor on different occasions. This occurs mainly after the External Controller resets the ST7 in a specific mode called ICC Mode. This is done by driving a specific signal sequence on ICC lines while the $\overline{\text{RESET}}$ pin is pulled low. For devices featuring a Debug Module, the ICC Monitor can also be called after an external abort signal on one of the ICC lines, or after a hardware breakpoint (see Section 9 IN-CIRCUIT DEBUGGING (ICD) for more details).

The System Memory also contains a dedicated set of interrupt vectors for ST internal use. When the ST7 enters ICC Mode, this dedicated set of vectors is valid and not those of the USER Mode, i.e. the reset vector refers to the ICC Monitor and not to the user application.

On the ST7s featuring the basic ICC Monitor, only 2 commands are available. As explained in Section 5 ICC COMMAND LIBRARY, it can be useful to make the ST7 understand more commands. To do so, the External Controller must upload an upgraded monitor (medium variant for example), write it into the ST7 RAM and then execute it (using the <Go> command which is always available).

On the ST7s featuring a Debug Module, the embedded monitor is the most advanced one, with all the commands, plus an automatic context saving at the beginning of the ICC Session.

Refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual.

8 BUILDING AN ICC INTERFACE TO A PC PARALLEL PORT

This example shows that ICC protocol can easily be used to communicate between an ST7 and a standard Personal Computer, even if the PC is slower than the MCU or is slowed down by multiprocessing.

8.1 HARDWARE INTERFACE

8.1.1 Basic PC Interface Board

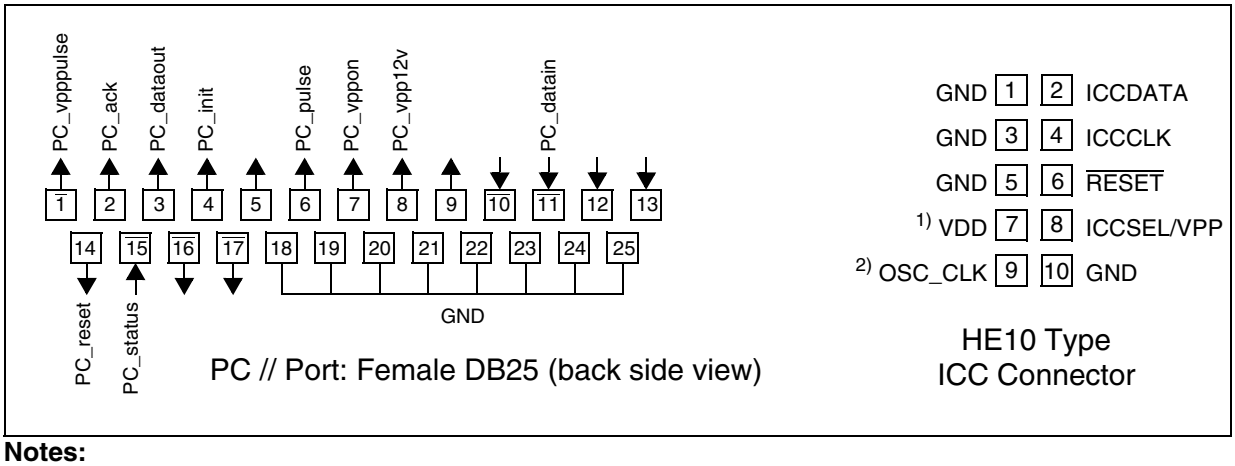
The ICC pins cannot be directly connected to a PC parallel port connector due to ICC protocol requirements, e.g. on the External Controller side, bidirectional I/Os can be connected to pull-up resistors. So an additional hardware interface is required. This hardware is also necessary to meet the time constraints imposed by the physical protocol.

Figure 15 "Example of PC Parallel Port to ICC Connection Interface Board" on page 29 shows an example of a circuit schematic diagram for such an interface board when the application is supplied with a voltage between 4.5V and 5.5V. This schematic can easily be implemented with a few ST components such as the 74HCT (or 74FCT) and 74ACT series (TTL input, CMOS output), ST662A 12V charge pump and transistors. Make sure that these components are compatible with the ST7 logic levels (CMOS input and output).

The interface between the External Controller and the ST7 application board is made of up to 7 lines. Figure 14 recommends a standard implementation of these lines with an HE10 type connector. This connector is used by ST Microelectronics software development tools and most of the third party tools.

The interface between the External Controller and the PC is made of a standard parallel port interface. Up to 8 output lines, 2 input lines and 8 ground lines are used as shown in Figure 14.

Figure 14. Recommended Standard ICC Connectors

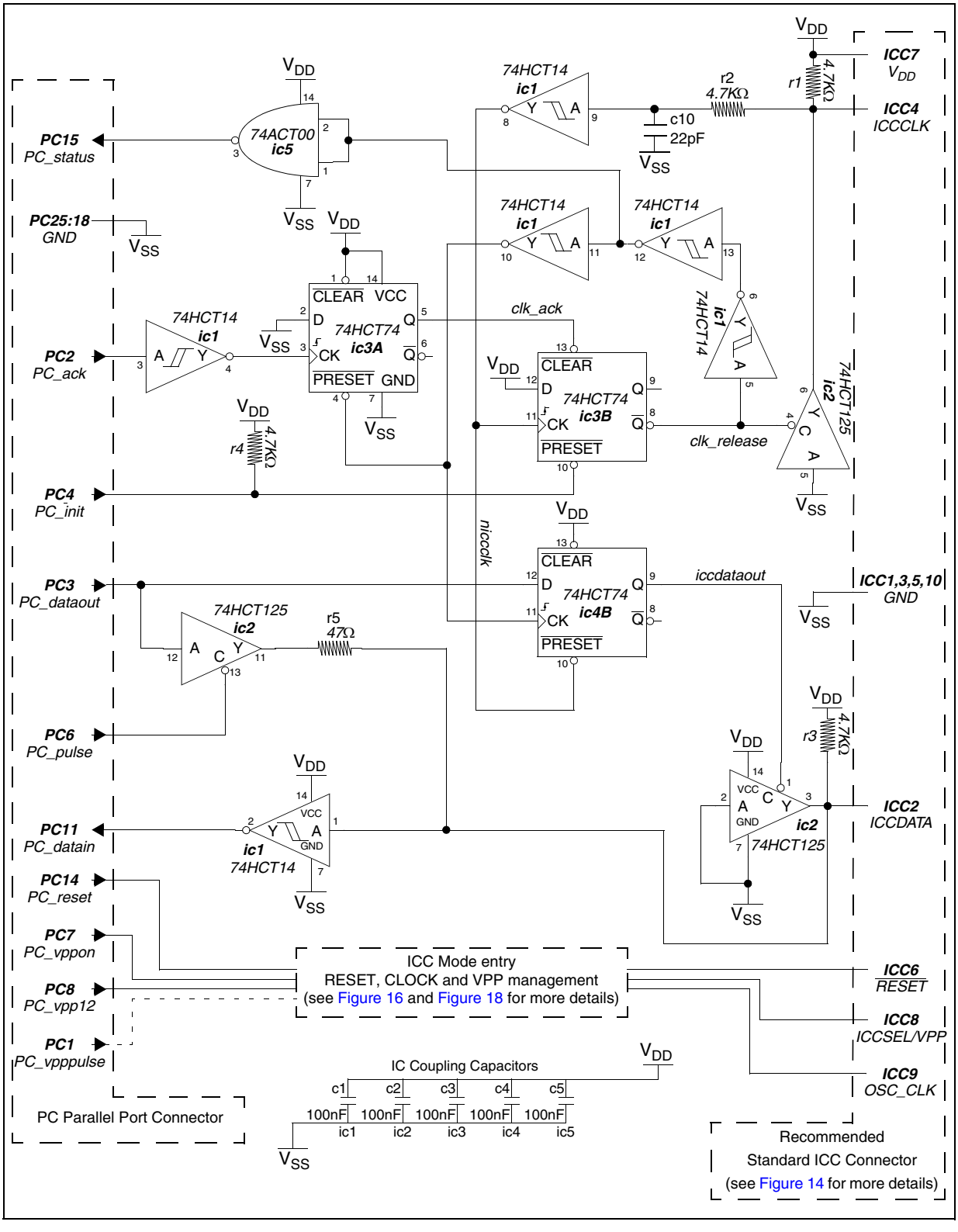


Notes:

1. For the board example defined in this section, the VDD line (7) is used to supply the Controller board. However, this line must be connected to the application VDD with most of STMicroelectronics development tools (for communication level adaptation for example); refer to ST programming tool manual.

2. The OSC_CLK line (9) has to be connected to the OSC1 or OSCIN pin of the ST7 when the clock is not available in the application or if the external clock option is programmed in the option byte.

Figure 15. Example of PC Parallel Port to ICC Connection Interface Board



8.1.2 Board ICC Mode Entry

As described in Section 3 ENTERING ICC Mode, there are two different ICC Mode entry methods. Consequently two hardware implementation examples are proposed:

- Section 8.1.2.1 describes a hardware which supports only the controlled window ICC Mode entry (the associated schematic is shown in Figure 16).
- Section 8.1.2.2 shows a hardware which support both mode entry sequences (controlled and fixed $256T_{CPU}$ window (the associated schematic is shown in Figure 18).

Both hardware proposals provide the programming voltage capability (V_{PP}) for dual voltage HDFlash. They are also compatible with ICC STMicroelectronics development tool software.

Please also refer to the STICK manufacturing files for information on building hardware for devices which supports controlled window ICC mode entry

8.1.2.1 Hardware for Controlled Window ICC Mode Entry

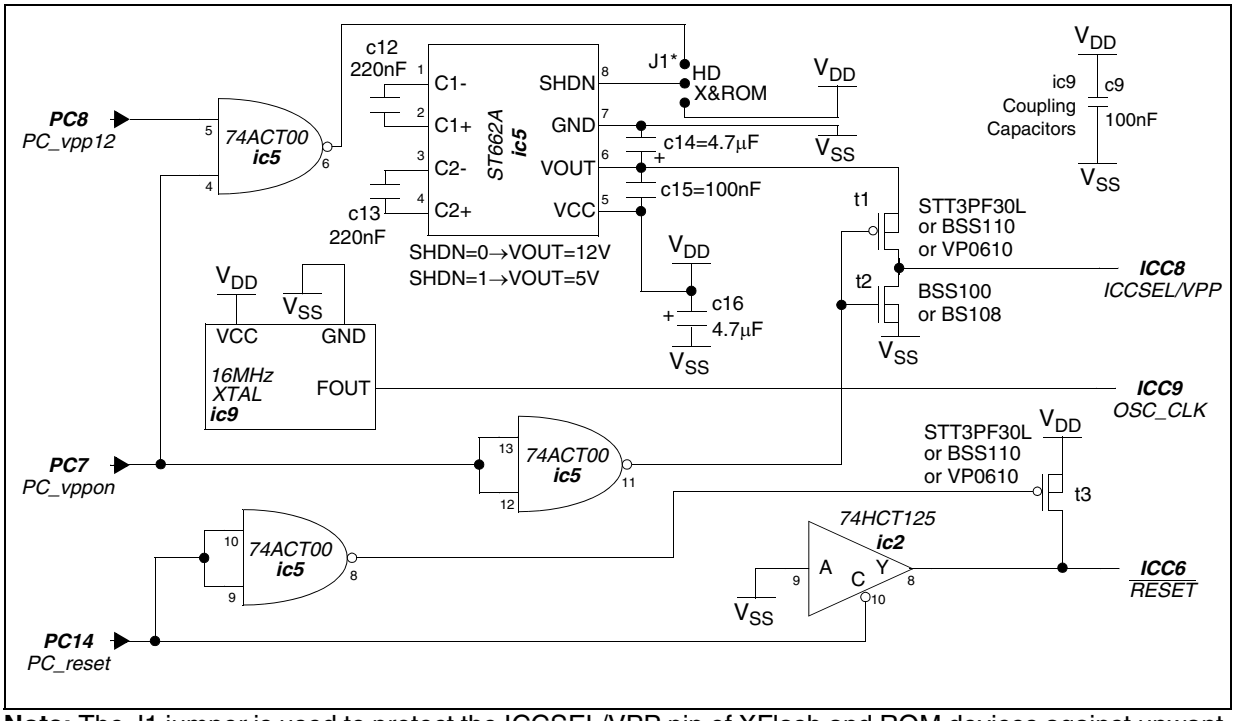
The schematic diagram is shown in Figure 16.

The PC uses the PC_pulse signal to drive asynchronous pulses on ICCDATA. This signal is used when entering ICC Mode (see section 3.1 on page 10) and also for generating an ABORT external interrupt (see section 9.1 on page 39).

The PC uses the PC_vpp12 and PC_vppon to drive three different voltages on the V_{PP} pin.

PC_vpp12	PC_vppon	V_{PP}	Remarks
X	0	0V	Applied during ICC Mode entry sequence when ICCSEL pin is used for counter reset
0	1	5V	Standard voltage to be applied during ICC Mode
1	1	12V	Programming voltage for dual voltage HDFlash

Figure 16. Hardware Implementation Example for Controlled Window ICC Mode Entry



Note: The J1 jumper is used to protect the ICCSEL/VPP pin of XFlash and ROM devices against unwanted 12V. The ST662A input voltage must be sufficient to ensure a VOUT of 11.7V min.

8.1.2.2 Hardware Compatible with all ICC Mode Entry Types

The schematic diagram is shown in Figure 18.

When the 256 t_{CPU} fixed window ICC Mode entry is used, the rising time of the ST7 $\overline{\text{RESET}}$ pin fixes the start time of the fixed window. Therefore the interface hardware must ensure a sharp edge on this $\overline{\text{RESET}}$ pin, while avoiding any conflict with the output capability of this $\overline{\text{RESET}}$ pin. This is why the proposed hardware is based on a momentary push-pull output (using transistor t3), before switching back in high-impedance mode (after 29µs; see Figure 17 for timing analysis). Note also that this hardware implies no capacitance constraint on the $\overline{\text{RESET}}$ pin. ICCDATA is kept low to let the ST7 recognize ICC Mode by using the same circuitry as the Controlled Window ICC Mode Entry.

The hardware described in Figure 18 can be also used to enter ICC Mode with a controlled window keeping the PC_vpppulse signal low.

The proposed hardware allows different working modes summarized in the table below.

PC_vpp12	PC_vppon	PC_vpppulse	V_{PP}	Remarks
X	0	X	0V	Pull-down by NMOS transistor
0	1	0	5V	Pull-up by PMOS transistor
0	1	1	0V / 5V	Pulsed ICCSEL/V_{PP} pin for fixed 256 t_{CPU} window ICC Mode entry

PC_vpp12	PC_vppon	PC_vpppulse	V_{PP}	Remarks
1	1	X	12V	Programming voltage for dual voltage HDFlash

Figure 17. Pulse Counter Timings for ICC Mode Entry

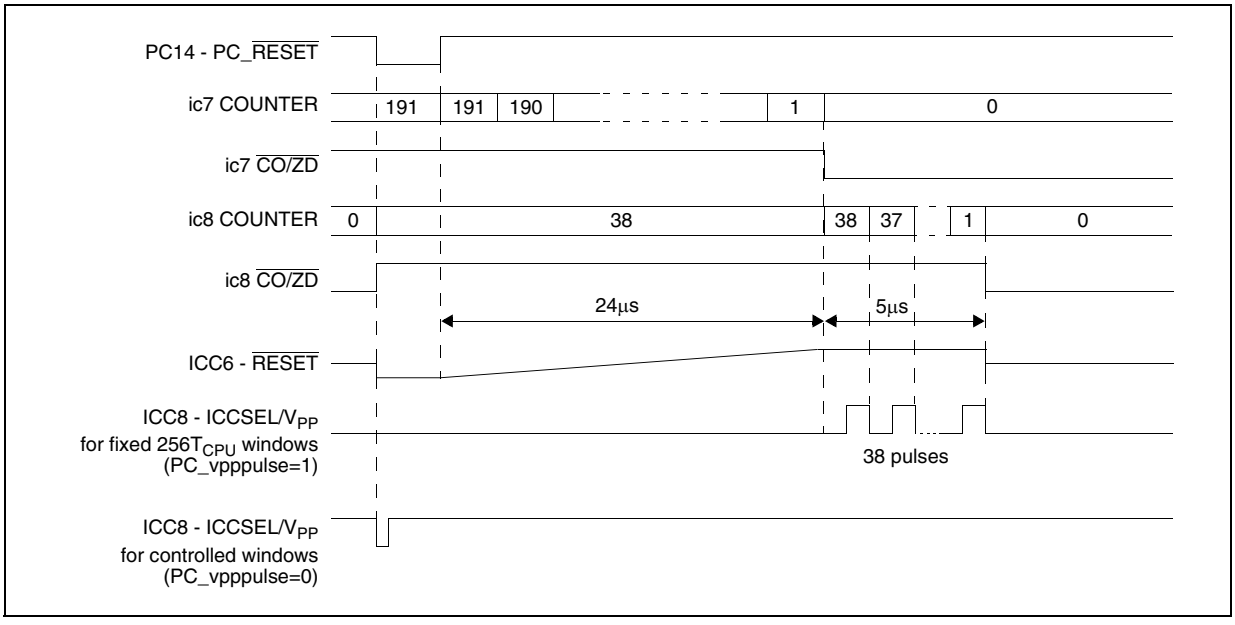
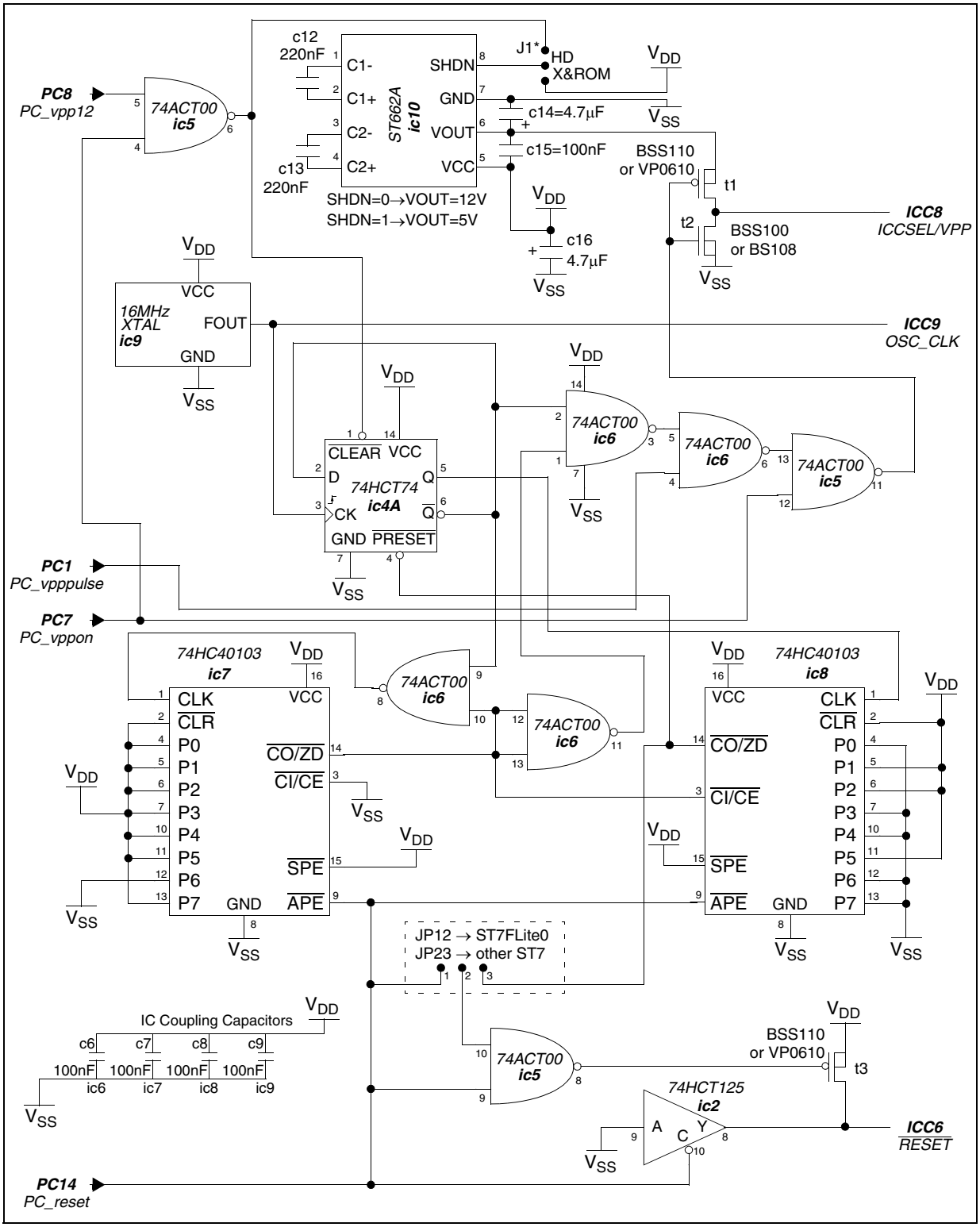


Figure 18. Hardware Implementation Example for all ICC Mode Entry Types



Note: The J1 jumper is used to protect the ICCSEL/VPP pin of XFlash and ROM devices against unwanted 12V.

8.1.3 Board Communication Interface

The following explanations refer to Figure 15 "Example of PC Parallel Port to ICC Connection Interface Board" on page 29. Two cases are distinguished:

- **When ICCCLK is used to reset the pulse counter when entering ICC Mode**
  During this phase, the ST7 is in RESET state and does not drive ICCCLK. The PC can therefore tie ICCCLK low by applying a pulse on PC_init and can release ICCCLK by applying a pulse on PC_ack. The PC can check the state of ICCCLK by reading PC_status because the ST7 is not driving ICCCLK.
- **When ICCCLK is used as clock to sample ICCDATA**
  The ST7 initializes communication by applying a falling edge on ICCCLK; this falling edge presets the ic3b latch. Consequently, the ic3B latch ties/keeps ICCCLK low through the tri-state buffer. The PC is informed of this communication start by reading PC_status. Following this stage, four configurations exist:
  – The PC is receiving bits: First, the PC must make sure that PC_dataout and PC_pulse are high in order to release ICCDATA. Then the PC reads the data on PC_datain and then it releases ICCLCK by applying a pulse on PC_ack to get the next bit.
  – The PC is sending bits: The PC prepares the next bit communication on PC_dataout and then it releases ICCLCK by applying a pulse on PC_ack to be ready for next bit to be sent.
  – The PC is receiving the last bit of a byte and then sends a byte: First, the PC reads the data on PC_datain, then it prepares the next bit on PC_dataout. Finally it releases ICCLCK by applying a pulse on PC_ack to be ready for next bit to be sent.
  – The PC is sending the last bit of a byte and then receives a byte: The PC gets the ICCDATA ready for received byte by setting PC_dataout high. Then it releases ICCLCK by applying a pulse on PC_ack to be ready for next bit to be received.

As long as ICCCLK is in high state, the board releases ICCDATA. PC_data_out is sampled on ICCCLK falling edge. The PC sets PC_dataout high either for a receive or a “send one” operation. It ties PC_dataout low only for a “send zero” operation when ICCCLK is low.

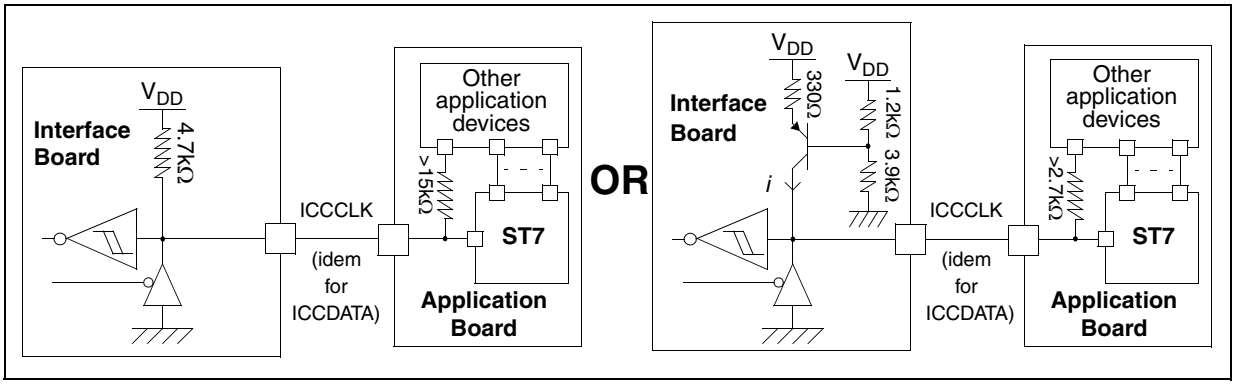
For a receive operation, the PC reads the bit on PC_data_in. The 3-inverter delay line prevents a glitch from occurring on ICCDATA when the transmission direction switches from a send to a receive operation.

8.1.4 ICCCLK and ICCDATA pull-ups for open-drain communication

In the schematic diagram (Figure 15 "Example of PC Parallel Port to ICC Connection Interface Board" on page 29), the two pull-up resistors (r1 and r3) can be replaced by current sources to guarantee the correct voltage levels when the application board contains serial resistors as suggested in the device datasheet (see Figure 19).

If the application board does not contain a serial resistor or if the “Other application devices” do not force the ICC lines (floating or pull-up input, open-drain output released...), the interface board will only need a pull-up resistor.

Figure 19. PC to ICC Interface Board - ICCCLK and ICCDATA Signals



Important Note: To avoid issues on ICCDATA and ICCCLK high level drives, the application should use these two ST7 pins as outputs. So no serial resistor is needed to isolate the application.

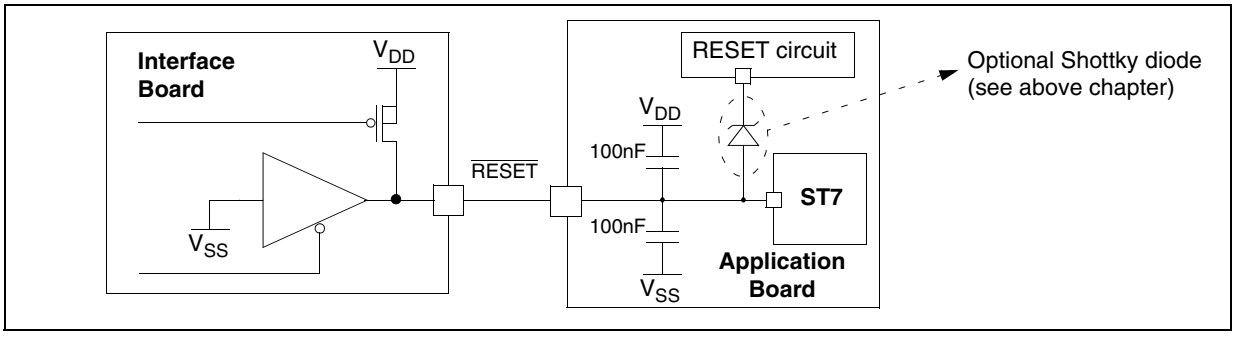
8.1.5 RESET Pull-ups for Open-drain Communication

In the schematic diagram (Figure 18), when the jumper is set between 2 and 3, the RESET signal output from the interface board is temporary configured in push-pull output to guarantee accurate timings for 256 t_{CPU} fixed window ICC Mode Entry.

During ICC Session, the programming tool must control the RESET pin. This can lead to conflicts between the programming tool and an application reset circuitry driving more than 5mA at high level (push pull output or pull-up resistor <1K). A Schottky diode can be used in that case to isolate the application RESET circuit. When using a classical RC network with R>1K or a power on RESET management IC with open drain output, no additional components are needed (see Figure 20).

If the application RESET circuit drives more than 5mA, the user must ensure that no external RESET is generated by the application during the ICC Session.

Figure 20. PC to ICC Interface Board - RESET Signal



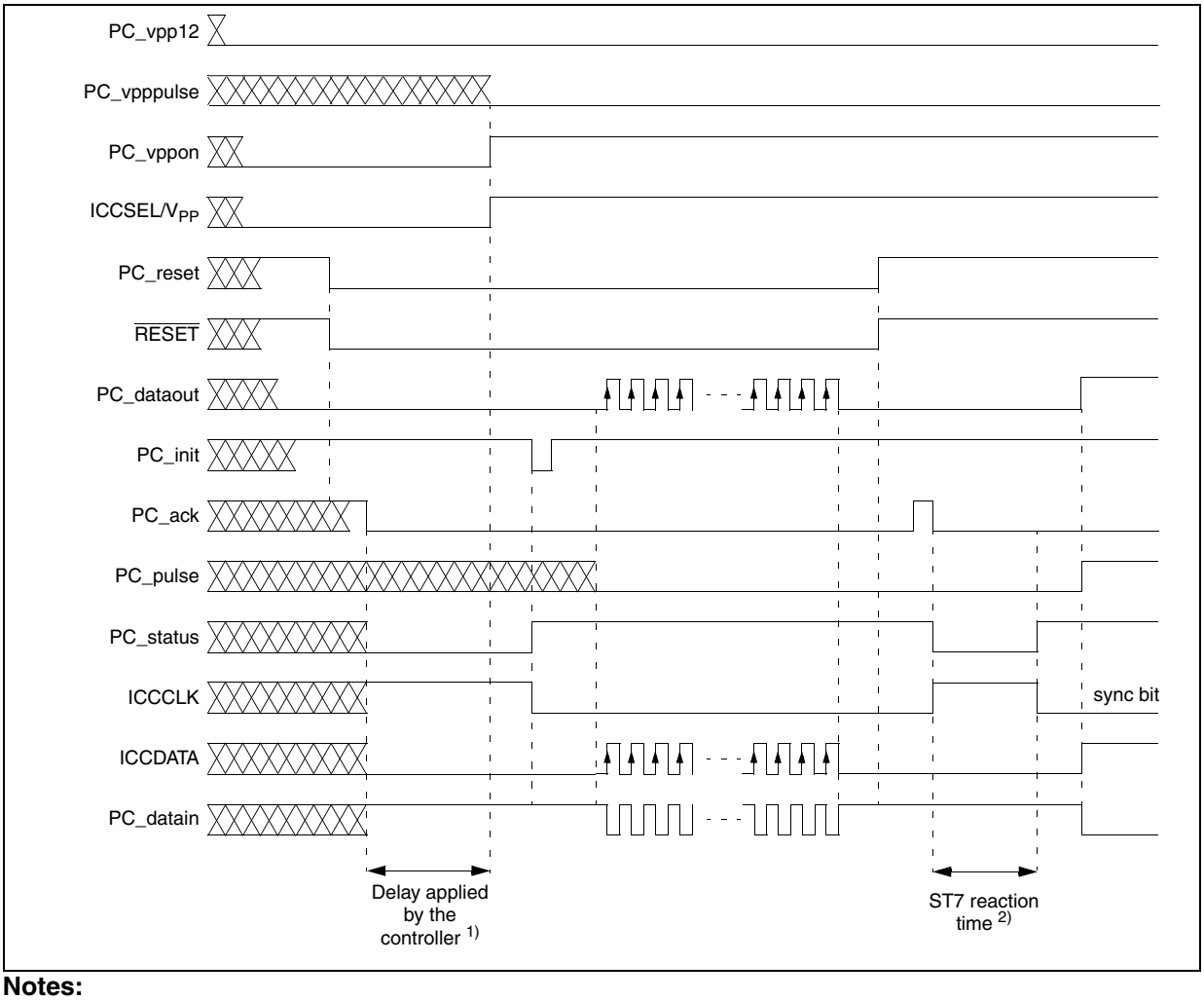
Important Note: The user must ensure that the external RESET is not driven (to V_{DD} or V_{SS}) by the application during ICC Session. See above section for more details.

8.2 ICC SIGNALS AND PC SOFTWARE INTERFACE EXAMPLE

This section gives chronogram examples of the ICC interface from the PC side, using the ICC connection interface board shown in Figure 15. To generate and manage the PC_xxx signals described in Figure 15, you can use the MSDOS PC software driver routines (running on Windows95/98) proposed in APPENDIX 2 on page 59.

Note: The following chronograms assume that the PC side is much slower than the ST7.

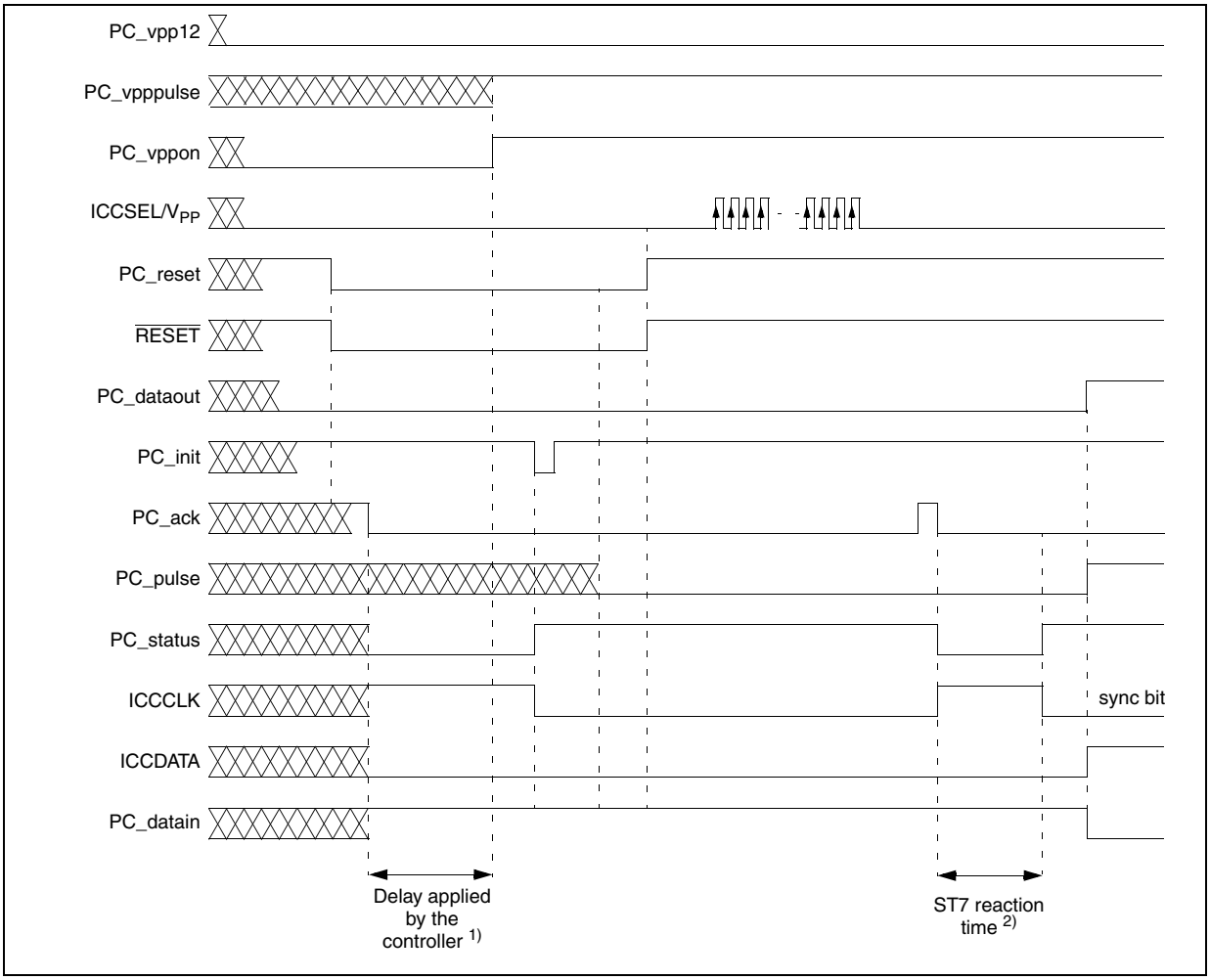
Figure 21. Controlled Window ICC RESET Phase Timings



Notes:

1. Delay applied by the controller to be sure that the RESET is taken into account by the device. This delay is needed due to the embedded RESET filter.
2. The controller waits for PC_status line high. This event means that ST7 as released ICCCLK to 1 and then force it back to 0.

Figure 22. Fixed 256 t_{CPU} Window ICC RESET Phase Timings



Notes:

1. Delay applied by the controller to be sure that the RESET is taken into account by the device. This delay is needed due to the embedded RESET filter.
2. The controller waits for PC_status line high. This event means that ST7 as released ICCCLK to 1 and then force it back to 0.

Figure 23. ICC Send Byte Timings (PC ⇒ ST7)

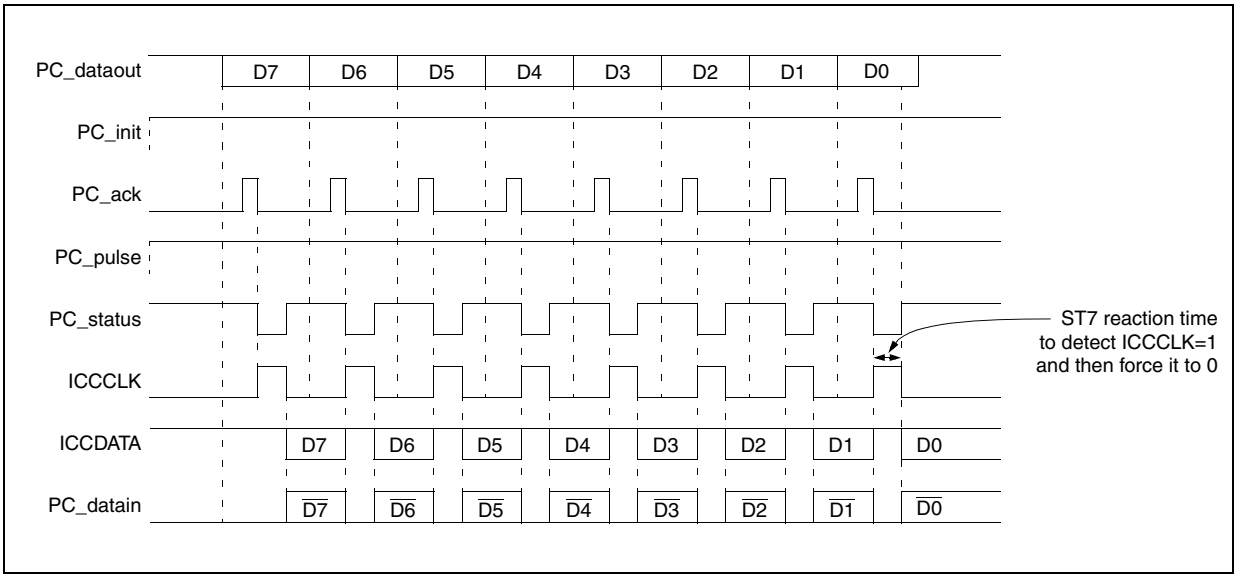
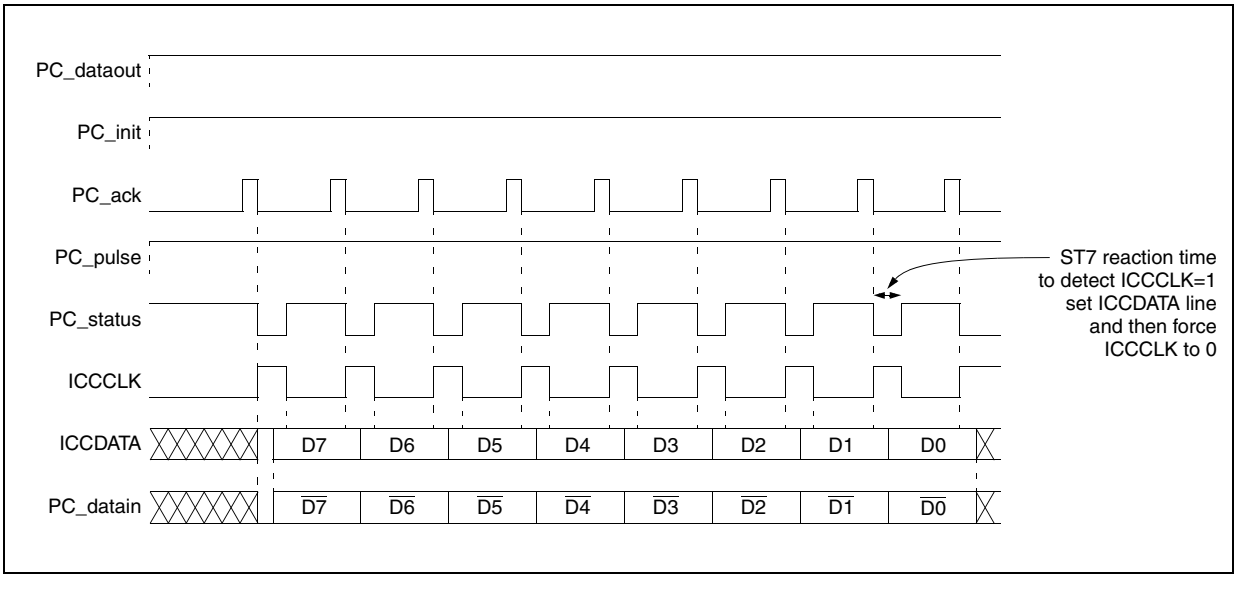


Figure 24. ICC Receive Byte Timings (ST7 ⇒ PC)



9 IN-CIRCUIT DEBUGGING (ICD)

In-Circuit Debugging is the ability to identify and eliminate anomalies in a program stored in a FLASH device using the ICC protocol. This feature is used to design low-cost emulators. ICD can be managed in two different ways depending if the Debug Module (DM) is included in the ST7 device (refer to the implemented ICC features table in the FLASH Programming Quick Reference Manual). This section describes how to easily manage the ICD.

CAUTIONS:

1. As this low-cost emulator controls the ST7 using the ICC protocol, it is not possible to use the two ICC pins (ICCCLK and ICCDATA) in the application.

2. When debugging using ICD, care must be taken in the application software to not access the Debug Module registers and the ICCCLK and ICCDATA I/O control/data registers (masking the bit for example). Otherwise the emulator can be lost.

9.1 ST7 DEVICES WITH DEBUG MODULE

Some ST7 devices have an embedded Debug Module. This dedicated hardware and the ICC capability can be used for easy management of:

- advanced hardware breakpoints on instruction, address or data
- instruction steps
- abort of the running application

Refer to the “Debug Module Reference Manual” for more details.

9.1.1 ICC Monitor Call on DM Request

In addition to the dedicated ICC RESET sequence (described in Section 3 ENTERING ICC Mode), when the Debug Module is included in the device, there are two other ways to execute the ICC Monitor and they are described below. With or without the Debug Module, a third way to execute the ICC Monitor is a “Software Breakpoint” managed with a TRAP instruction (see section 9.2 on page 46 for more details). These three methods are available only in USER Mode.

“Abort”: If the Debug Module has been properly configured, it generates a non-maskable interrupt request as soon as the External Controller drives a falling edge on the ICCDATA pin. The interrupt vector corresponding to this interrupt request is the user software interrupt (TRAP) vector, so the ST7 starts an ICC Session provided the TRAP vector refers to the ICC Monitor.

“Hardware Breakpoint”: If the Debug Module has been properly configured, it generates a non-maskable interrupt (NMI) request when the ST7 complies with hardware breakpoint conditions. The interrupt vector corresponding to this interrupt request is the user software interrupt (TRAP) vector, so the ST7 starts an ICC Session provided the TRAP vector refers to the ICC Monitor.

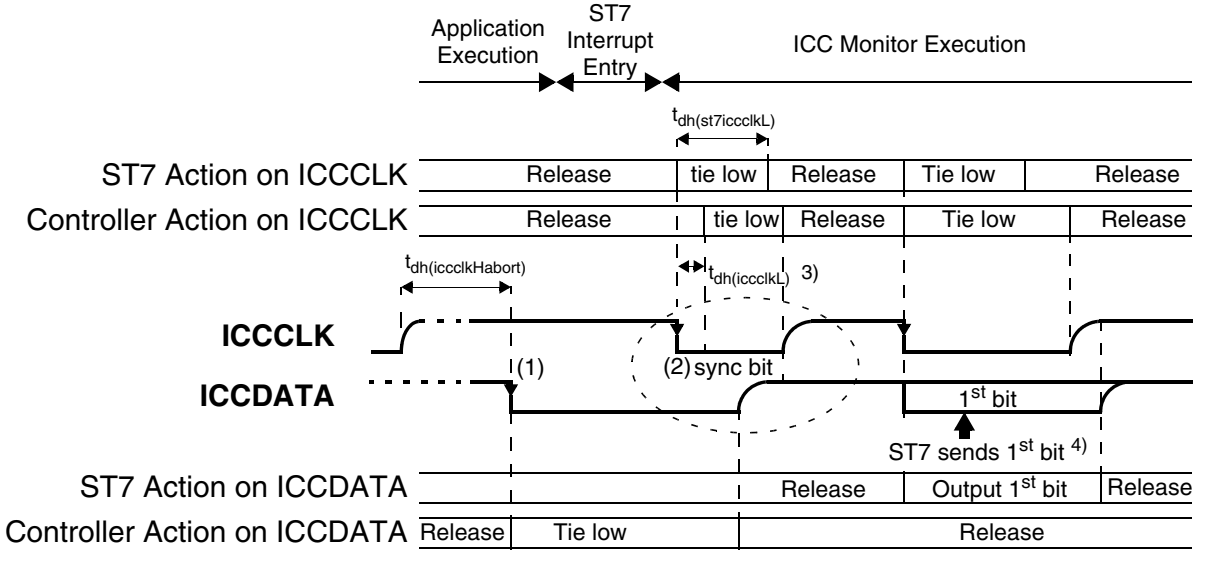
Note: Refer to PC software routine examples defined in APPENDIX 2 on page 59 to manage the ICD Abort and Breakpoint capabilities with the ICC connection interface board described in section 8 on page 28.

9.1.1.1 ICC Monitor called by an ICD Abort

When this functionality is enabled in USER Mode (see Section 9.1.2 and Section 9.1.3 for more details), the ST7 jumps into the ICC Monitor when the External Controller drives a falling edge on ICCDATA (Figure 25). As long as the External Controller does not detect an ST7 reaction on ICCCLK, it keeps ICCDATA low.

As the ST7 starts with the sync bit, it drives a falling edge on ICCCLK. This causes the External Controller to tie ICCCLK low. Then the External Controller releases ICCDATA. Finally, both devices release ICCCLK, and the system is ready for the first bit exchange.

Figure 25. Start of an ICC Session when the ICC Monitor is called by an DM Abort



Symbol	Description	Min	Max	Unit
$t_{dh(st7icccIkL)}$	Period that the ST7 ties ICCCLK low	16		t_{CPU}
$t_{dh(icccIkHabort)}$	Period between the release of ICCCLK by the ST7 and an ABORT	21		
$t_{dh(icccIkL)}$	Period during which the External Controller must detect ICCCLK low and tie it to low		$t_{dh(st7icccIkL)}$	

Notes:

1. Falling edge generating a non-maskable interrupt for the ST7.

2. Falling edge indicating the start of the ICC Monitor. When ICCCLK is low, ICCDATA is not taken into consideration by the ST7 device.

3. The External Controller must tie the ICCCLK line low before a delay of $t_{dh(icccIkL)}$. This delay corresponds to the minimum time needed by the ST7 to release the ICCCLK line.

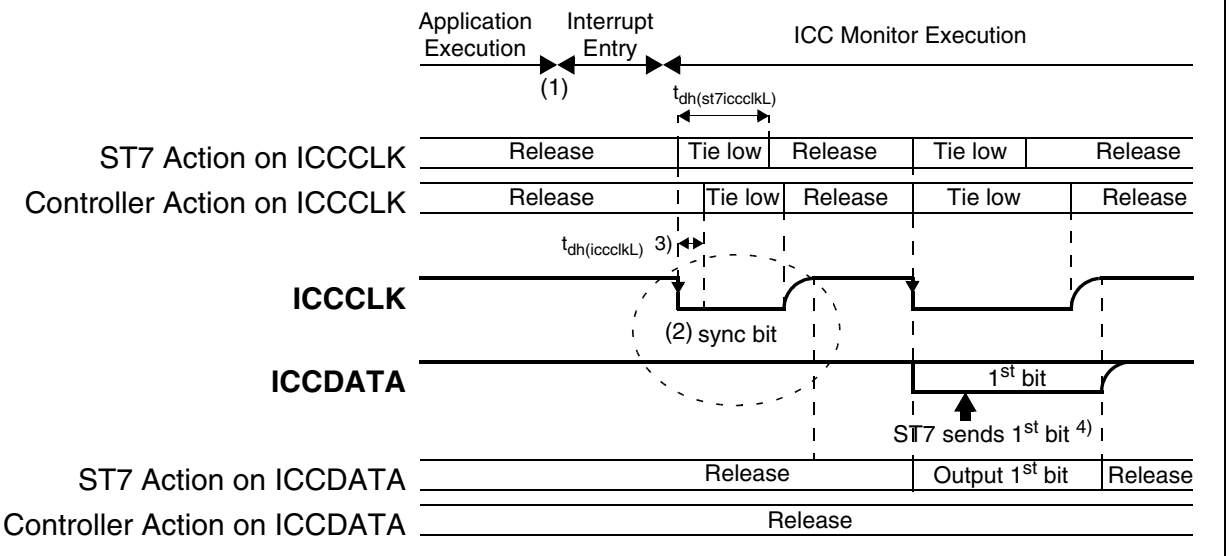
4. The ST7 first sends 6 data bytes to the External Controller for automatic context saving.

9.1.1.2 ICC Monitor called by an DM Hardware Breakpoint

When this function is enabled in USER Mode (see Section 9.1.2 and Section 9.1.3 for more details), the ST7 jumps into the ICC Monitor after an internal breakpoint event. As it starts with the sync bit, it drives a falling edge on ICCCLK (see Figure 26). The External Controller detects this falling edge and knows that the ST7 has entered the ICC Monitor. It ties ICCCLK low.

Then both devices release ICCCLK and the system is ready for the first bit exchange.

Figure 26. Start of ICC Session when the ICC Monitor is called by an DM H/W Breakpoint



Symbol	Description	Min	Max	Unit
$t_{dh(st7icccklL)}$	Period that the ST7 ties ICCCLK low	16		t_{CPU}
$t_{dh(icccklL)}$	Period during which the External Controller must detect ICCCLK low and tie it to low		$t_{dh(st7icccklL)}$	

Notes:

1. Event generating a non-maskable interrupt inside the ST7.

2. Falling edge indicating the start of the ICC Monitor. When ICCCLK is low, ICCDATA is not taken into consideration by the ST7 device.

3. The External Controller must tie the ICCCLK line low before a delay of $t_{dh(icccklL)}$. This delay corresponds to the minimum time needed by the ST7 to release the ICCCLK line.

4. The ST7 first sends 6 data bytes to the External Controller for automatic context saving.

9.1.1.3 ICC Monitor called simultaneously by two DM events or a TRAP

The ST7 jumps into the ICC Monitor after an internal event (software TRAP or hardware breakpoint), but at the same time the External Controller drives a falling edge on ICCDATA (abort). In this case, only one NMI is generated. As the first bit exchange is a dummy, this low level on ICCDATA is not taken into account, and the communication can be initiated as though it had not happened (see case above).

Note: The External Controller cannot know what caused the ICC Monitor to start only by reading the ICC signals. To obtain this information, the DMCSR register must be checked.

9.1.2 ICD Implementation with XFlash devices

The chapter describes how to implement the ICD capability with a debugger and an ST7 device including a Debug Module (with Advanced ICC Monitor) and XFlash.

In XFlash products, the System Memory is executable in USER Mode so the Advanced ICC Monitor does not need to be copied into the user XFlash area.

Before starting In-Circuit Debugging, the debugger must configure the ST7 option bytes using ICP. The watchdog must always be configured as software (hardware watchdog has to be emulated using the WDG control bit of the DMCR register, see the “Debug Module Reference Manual” for more details).

How to start ST7 ICD?

This procedure is divided into 3 steps and its result in terms of memory organization is shown in Figure 27.

– *Step 1*: Debugger replaces the user application RESET and TRAP vectors by the `ICC_mode` and `ICC_monitor_trap` addresses located in the System Memory keeping the original vector in the PC memory. These addresses (refer to the system memory addresses table in the FLASH Programming Quick Reference Manual) allow the debugger to manage user RESET and TRAP.

– *Step 2*: Using ICP (ICC RESET sequence), the debugger programs the patched user application code in the XFlash.

– *Step 3*: To start the debugging phase, the debugger resets the device in USER Mode. This operation will make the ST7 fetch the ICC Monitor. The flowchart shown in Figure 28 describes an example using the PC driver routines defined in APPENDIX 2 on page 59.

Figure 27. ICD Memory Organization for XFlash Products

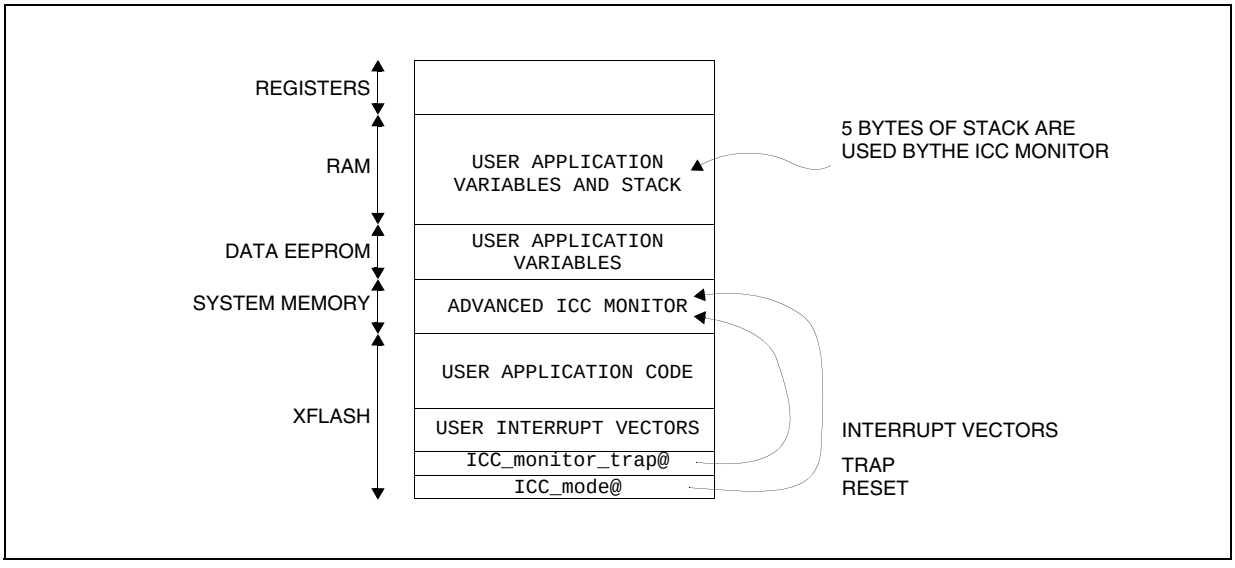
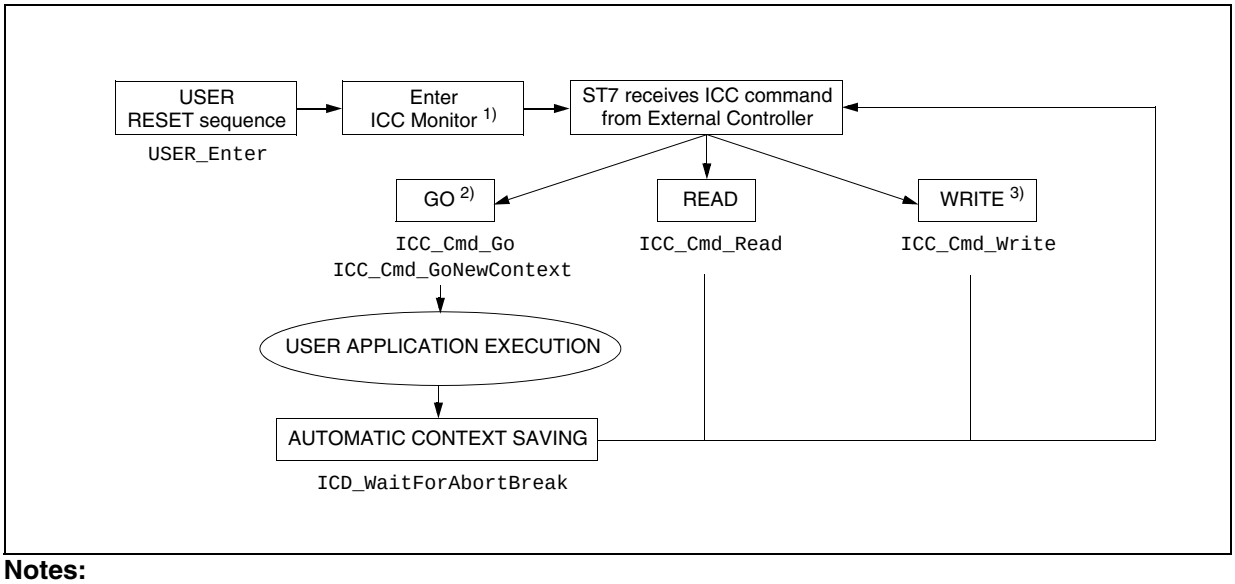


Figure 28. ICD Implementation Flowchart



Notes:

1. “RESET” and “TRAP” vectors contain ICC Monitor entry points located in System Memory (see "How to start ST7 ICD?" for more details).

2. The <go> command is executed after a RESET sequence. Following a Debug Module “Abort” or “Breakpoint”, a <go new context> command is executed. See section 5 on page 19 for more details.

3. For example, the <write memory> command can be used to setup the Debug Module (Abort / Breakpoints) by writing to its registers or to write to the stack the address to where the next <go> command must jump.

How does the debugger control the ST7 ICD after the USER Mode RESET?

This procedure is divided into 3 steps.

– *Step 1*: The debugger configures the Debug Module using ICC commands. Breakpoints and stepping can be set according to the user requirements. At least the abort function has to be enabled.

– *Step 2*: Using ICC commands, the debugger writes the user RESET address (previously stored in the PC memory) in the PCH and PCL stack locations and the CCR reset value (111x1xxxb) in the CCR stack location.

– *Step 3*: Start the user application using the ICC <go> command.

Note: When the ST7 executes the ICC Monitor, the debugger can read or write the memory space (peripheral registers, RAM or FLASH) using ICC commands. The CPU registers can be accessed and modified through the stack and the “Automatic Context Saving”.

How does the debugger manage the abort and breakpoints?

When an abort is initiated by the debugger or a software or hardware breakpoint occurs, the ST7 stops the user application and enters the ICC Monitor program through the patched TRAP vector (refer to section 9.1.1 on page 39 for more details). Following that, the debugger performs 3 steps.

– *Step 1*: The debugger receives 6 ICC automatic context saving bytes and stores them into the PC memory (see section 5.4 on page 22).

– *Step 2*: The debugger determines why the user application has stopped by reading the DMCSR register bits. The software breakpoint (TRAP instruction) is detected if none of the DMCSR status bits are set.

– *Step 3*: The debugger is ready to receive the user debug commands thanks to a Graphical User Interface (GUI). If this command is “program resume”, the debugger makes the user application resume using the ICC <go new context> command (the 6 bytes to be sent are recovered from the ones previously stored in the PC memory).

Caution: When an application RESET occurs while the External Controller is waiting for a breakpoint, both sides wait for an ICC byte reception (the ST7 waits for an ICC command and the External Controller waits for 6 automatic context saving bytes). In this case, the debugger and the ST7 resynchronize after getting 6 times FFh on ICCDATA (as FFh is an incorrect command for ST7, it waits for the next one).

How does the debugger manage the user application RESET?

The user application RESET can be managed by the ICD if the application makes sure that this RESET never occurs while the ICC Monitor is executed.

– For the internal watchdog RESET source, this can be done by configuring the software watchdog (see the “Debug Module Reference Manual” for more details).

– For the other sources, a RESET is not allowed while the ICC Monitor is executed.

9.1.3 ICD Implementation with HDFlash Devices

This section describes how to implement the ICD capability with a debugger and an ST7 device including a Debug Module (with Basic ICC) and HDFlash.

In HDFlash products, the System Memory is not accessible in USER Mode so the Advanced ICC Monitor must be copied into the user HDFlash area. Consequently when debugging HDFlash products, the memory size available for the user application is reduced by approximately 256-bytes.

The flowchart shown in Figure 28 "ICD Implementation Flowchart" on page 43 describes an ICD implementation example using PC driver routines defined in APPENDIX 2 on page 59.

How to start ST7 ICD?

This procedure is divided into 4 steps and its result in terms of memory organization is shown in Figure 29.

– *Step 1*: The debugger adds the “Advanced ICC Monitor” to the user application code.

– *Step 2*: The debugger replaces the user application RESET and TRAP vectors by the `ICC_mode` and `ICC_monitor_trap` addresses located in the advanced ICC Monitor keeping the original vector in the PC memory.

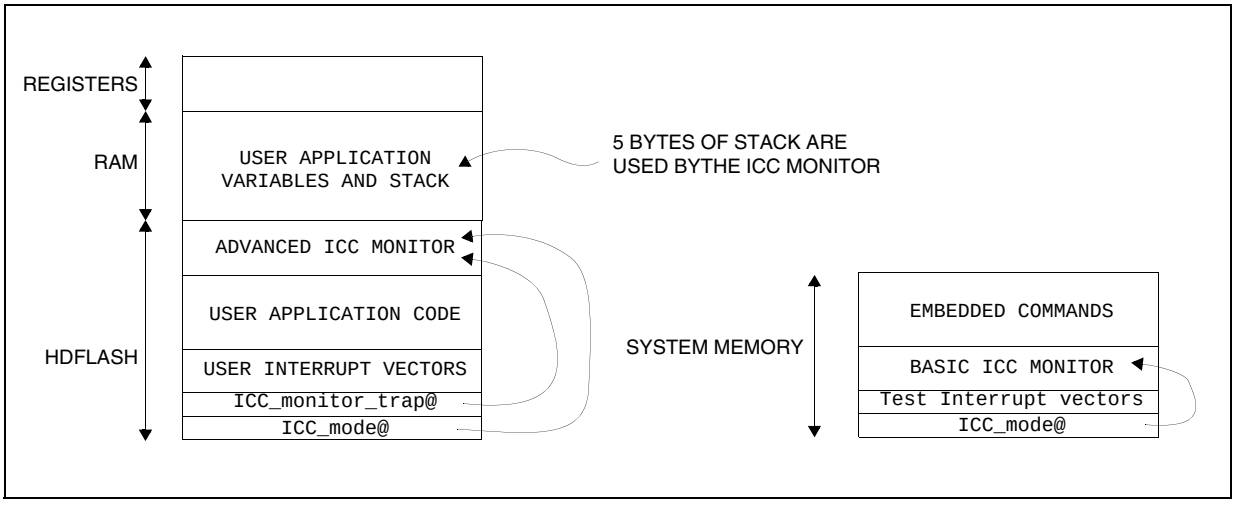
– *Step 3*: Using ICP (ICC RESET sequence), the debugger programs the patched user application code and ICC Monitor in the HDFlash.

– *Step 4*: To start the debugging phase, the debugger resets the device in USER Mode. This operation will make the ST7 fetch the ICC Monitor.

Debugging with ICD

For information on how the debugger controls the ST7 ICD after the USER Mode RESET, how it manages the abort and breakpoints and how it manages the user application RESET, please refer to Section 9.1.2, which applies both to HDFlash and XFlash.

Figure 29. ICD Memory Organization for HDFlash Products



9.2 ST7 DEVICES WITHOUT DEBUG MODULE

If the ST7 device does not include a Debug Module but has XFlash, the ICC capability handles a simple form of ICD with only software breakpoints on instruction.

The ICD implementation procedure is the same as the one described in section 9.1.3 on page 45 except the fact that the only way to stop the user application is a software breakpoint or a device RESET.

Software Breakpoints

To make a software breakpoint, the instruction where the ST7 must stop is patched with a TRAP instruction to enter the ICC Monitor. As this patch must be done byte by byte, this ICD can be implemented only in devices that support byte by byte erasing/programming (i.e. XFlash).

Note: Software breakpoints can be implemented in a debugger developed for ST7 devices with a Debug Module.

9.3 ICD GENERAL LIMITATIONS

This section describes the limitations for an ICD implementation.

- ICCCLK and ICCDATA pins are reserved and cannot be used by the application. Therefore, in the user application software, associated register bits in the port registers MUST always be masked to avoid conflict with the debugger.
- Real time emulation is not available for RESET and TRAP events because the associated vectors are replaced with the ICC Monitor vectors in the user application. The user RESET and TRAP routines are executed following a debugger action.
- The hardware watchdog option can not be used because the time spent in the ICC Monitor is unpredictable. This function is emulated by the debugger with a software watchdog.
- User application RESET source is not supported by the ICD unless the application guarantees that the external RESET never happens while the ICC Monitor is running.
- For HDFlash devices, the ICC Monitor must be copied in the user FLASH memory area because the System Memory is not accessible in USER Mode. The memory size available for the user application is then reduced.
- Each HDFlash device provides only limited ICD capabilities because its FLASH memory can support only limited programming/erasing cycles.

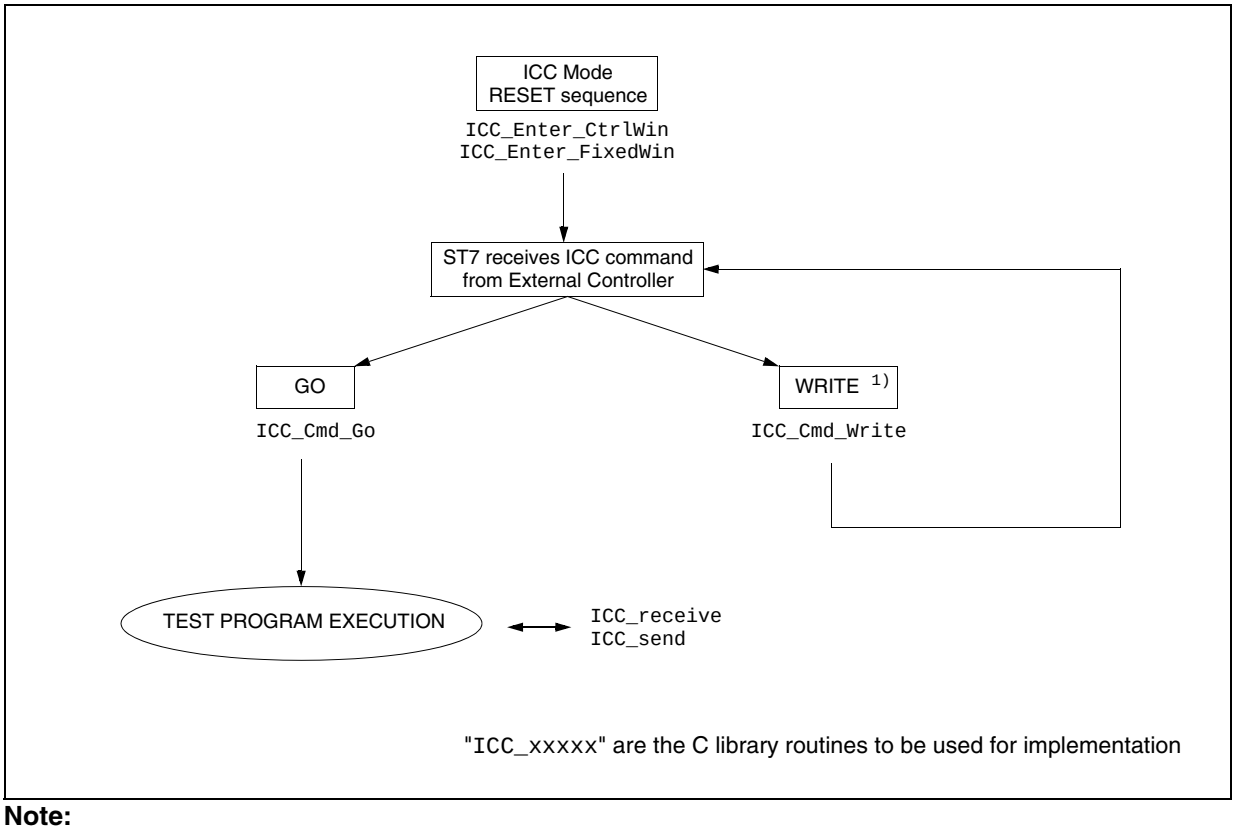
10 IN-CIRCUIT TESTING (ICT)

10.1 OVERVIEW

In-Circuit Testing (ICT) is a flexible method of performing production test routines that can be easily modified and extended without affecting the content of the FLASH program memory. This method uses the ICC protocol to download and execute the test program.

The flowchart shown in Figure 30 describes an ICT implementation using PC driver routines defined in APPENDIX 2 on page 59. This implementation is valid for all ST7 devices which include ICC as only the basic ICC version is required to execute this program.

Figure 30. ICT Implementation Flowchart



Note:

1. For example the “write” command can be used to write the test program in RAM and to write in the stack the address to where the next <go> command must jump.

10.2 ICT EXAMPLE: INTERRUPT VECTOR 8-BIT CHECKSUM

The following ICT example describes how to implement a test program using the ICC interface board (see section 8 on page 28) and the PC software driver described in APPENDIX 2 on page 59.

The purpose of this test program is to compute the 8-bit checksum of the user interrupt vector table (address FFF0h to FFFFh).

The PC software can be summarized with the following C code:

```
ICC_Enter_CtrlWin(Nb_Pulse);  // To make the ST7 enter ICC Mode
                              // or ICC_Enter_FixedWin();
ICC_Cmd_Write(); // => Download the test program into RAM from address 0090h
ICC_Cmd_Write(); // => Set the address 0090h at the top of the stack
ICC_Cmd_Go();    // Start the execution of the test program
CheckSum = ICC_receive();     // Get the result of the test program via ICC
```

The ST7 test program to be downloaded is shown below. The opcode bytes are downloaded sequentially into the RAM from 0090h to 00C7h using the `ICC_Cmd_Write()` driver routine. These bytes are marked in bold.

```
             st7/
             ;*****************************************************************
             ;
             ; TITLE:       ICTexample.asm
             ; AUTHOR:      CMG_MCD Application Team
             ; DESCRIPTION: ICT example (interrupt vector 8-bit checksum)
             ;*****************************************************************
             ;
                   TITLE    "ICTexample.asm"
                   BYTES

             ; ST7 Product ICC I/O Definition
             ;-----------------------------------------------

0008         PxDR  EQU $08  ; I/O port register definition for ICC management
0009         PxDDR EQU $09
0000               #DEFINE ICCCLK  0   ; Bit definitions for ICCCLK and
0000               #DEFINE ICCDATA 1   ; ICCDATA pin allocation

             ; Test program variable definition
             ;-----------------------------------------------

008F         PTRL EQU $8F         ; Two byte pointer to access the
008E         PTRH EQU $8E         ; interrupt vector table
008D         RES  EQU $8D         ; Checksum of the interrupt vector table

                   WORDS
                   SEGMENT  WORD AT 0090-00FF 'Ram'

             .Test_Program
             ;*****************************************************************

0090 A6FF          LD    A,#$FF          ; Set the interrupt vector start
0092 B78E          LD    PTRH,A          ; address (FF00h) in PTR pointer
0094 3F8F          CLR   PTRL
0096 AE10          LD    X,#16           ; Init X register as byte counter
0098 3F8D          CLR   RES             ; Init the checksum value to 00h
             .loop
009A 92D68E        LD    A,([PTRH.w],X)  ; Get a interrupt vector table byte
009D BB8D          ADD   A,RES           ; Compute the checksum
009F 5A            DEC   X               ; Next byte to read & check if FF00h
00A0 2AF8          JRPL  loop            ; address has been already reached
```

```
00A2 B68D          LD    A,RES           ; Get the checksum result
00A4 AD02          CALLR ICC_send_byte   ; and send it via ICC
               .end
00A6 20FE          JRT   end

               .ICC_send_byte              ; LOW LEVEL ICC PROTOCOL SUBROUTINE
               ;*********************************************************************
               ;
               ; ICC SEND subroutine used by the ST7 to send a byte on ICCDATA
               ; before start and after end of subroutine:
               ;        ICCCLK = 1 as floating input
               ;        ICCDATA as floating input
               ; before start of subroutine:
               ;        Acc A = byte to send
               ; after end of subroutine:
               ;         X = 0
               ; ICCCLK levels: 0 for 17 Tcpu
               ;                1 for 43 Tcpu minimum

00A8 AE08          LD    X,#$08   ; X = BitCounter
00AA 1209          BSET  PxDDR,#1 ; Define ICCDATA as open-drain output
               .icc_s1
00AC 1208          BSET  PxDR, #1 ; ICCDATA=1 because tying line to 0 is faster
00AE 49            RLC   A        ; Copy bit to transmit into carry flag
00AF 2502          JRC   icc_s2   ; Test bit to transmit
00B1 1308          BRES  PxDR, #1 ; If "0" set ICCDATA = 0
               .icc_s2
00B3 4D            TNZ   A        ; Dummy instructions to get a 2x3Tcpu delay
00B4 4D            TNZ   A        ; to insure the ICCDATA level
00B5 1108          BRES  PxDR ,#0 ; Because BSET/BRES ICCDATA may set ICCCLK DR
00B7 1009          BSET  PxDDR,#0 ; => ICCCLK output = 0
00B9 4D            TNZ   A        ; Dummy instructions to get a 4x3Tcpu delay
00BA 4D            TNZ   A        ; for External Controller reaction time
00BB 4D            TNZ   A
00BC 4D            TNZ   A
00BD 1109          BRES  PxDDR,#0 ; Release ICCCLK after 5 Tcpu
               .icc_s3
00BF 0108FD        BTJF  PxDR ,#0,icc_s3 ; Wait until ICCCLK = 1
00C2 5A            DEC   X     ; Ext. Controller already detected 1 (TTL input)
00C3 26E7          JRNE  icc_s1   ; Check if the whole byte is sent
00C5 1309          BRES  PxDDR,#1 ; If yes, define ICCDATA as floating input
00C7 81            RET
               ;*********************************************************************
               ;     END
```

APPENDIX 1 (ICC MONITOR ASSEMBLER CODE)

The “ICC CONFIGURATION”, “ASSEMBLER DEFINITIONS”, “INTERRUPT VECTORS” and “SEGMENT” definitions are defined for each ST7 product (bold part of the software).

Note: The ICD_Available and No_Basic_ICC constant identifiers define a scale of different ICC Monitors, from the basic to the advanced one (see Table 1, “ICC Monitor Commands,” on page 19).

```
st7/
;*****************************************************************************
; MODULE:       icc_drv.asm
; AUTHOR:       CMG_MCD Application Team
; UPDATE:       July 3rd, 2001
; DESCRIPTION: Generic In-Circuit Communication protocol
;              to be adapted to each device: fill in the product specific
;              informations (such as addresses, bit location, vectors...)
;              The example given in this file refers to the ST72F264 product.
;*****************************************************************************
; ICC Command Definitions
; ---------------------------------------------------------------------------
;  Command Name    | Command Definition
; ---------------------------------------------------------------------------
;  Write memory    | Write N memory bytes into ST7 addressing space
;                  | starting at MSB@:LSB@ address
;                  | MSbit first and MSbyte first
; ---------------------------------------------------------------------------
;  Go              | Perform an "IRET" instruction (die Id is stored at 0080h)
; ---------------------------------------------------------------------------
;  Go new context  | a) Fill in the 2 RAM addresses used by this
;                  |     program as an address buffer
;                  | b) Update stack pointer value
;                  | c) Store 2 bytes in the stack in order to fill in the
;                  |     A and CC registers at the next "IRET" instruction
;                  | d) Fill in the Y register
;                  | e) Perform an "IRET" instruction
; ---------------------------------------------------------------------------
;  Read memory     | Read N memory bytes from ST7 addressing space
;                  | starting at MSB@:LSB@ address
;                  | MSbit first and MSbyte first
; ---------------------------------------------------------------------------
;  Automatic       | ST7 send sequentially  Y, CC, A, 0081h, 0080h, S when
;  Context saving  | entering advanced ICC Monitor following an ABORT or BKPT
; ---------------------------------------------------------------------------

; If the ST7 features ICD, the <Go New Context> command is implemented
; instead of the <Go> command when entering the monitor with a breakpoint
; or an abort.

; ICC Command Protocol
; ---------------------------------------------------------------------------
;  Command Name    |  Coding   | Command Description
;                  |   (cmd)   |
; ---------------------------------------------------------------------------
;  Write memory    | 0xxxxxxx  | >cmd >MSB@ >LSB@ >N-1
;                  |           |      >D(@+N-1) >D(@+N-2) ... >D(@)
;  Go              | 10xxxxxx  | >cmd
;  Go New Context  | 10xxxxxx  | >cmd >D(buf@+1) >D(buf@) >A >CC >Y
;  Read memory     | 110xxxxx  | >cmd >MSB@ >LSB@ >N-1
;                  |           |      <D(@+N-1) <D(@+N-2) ... <D(@)
;  <No operation>  | 111xxxxx  | >cmd
```

```
; ---------------------------------------------------------------------------

; Legend:   N    is the number of bytes to send or receive
;           >    ST7 receives from External Controller
;           <    ST7 sends to External Controller
;           @    is the address stored in the address buffer
;           buf@ is the address of this buffer (the address of its LSByte)

; ICC used RAM space: 2 bytes (ADDRBUF and ADDRBUF+1)

; ICC used stack space:
;   * minimum 5 bytes (CC, A, X, PCH, PCL at interrupt call)
;   * no more than 5 bytes if the ST7 features ICD
;   * 2 bytes when executing a command (1 call)

;****************************************************************************

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; ICC CONFIGURATION
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

; ---------------------------------------------
  #DEFINE ICD_Available 1 ; |  1  |  0  |  0  |
  #DEFINE Not_Basic_ICC 1 ; |  1  |  1  |  0  |
; ---------------------------------------------
;  Write memory             |  x  |  x  |  x  | (x <=> command available)
;  Go                       |  x  |  x  |  x  |
;  Go new context           |  x  |     |     |
;  Read memory              |  x  |  x  |     |
;  Automatic context saving |  x  |     |     |
; ---------------------------------------------
;  Software size [bytes]    | 219 | 150 |  90 |
;  Vector   size [bytes]    |  36 |  36 |  36 |
; ---------------------------------------------

  TITLE "icc_drv.asm"

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; ASSEMBLER DEFINITIONS
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

  BYTES

; ST7 Product Definition ------------------------------------------------------
  #DEFINE RAMBEG      $80    ; RAM Start address definition

  #DEFINE DEVICE_ID   $8251  ; ST72F264 Die identifier + System Memory version

  #DEFINE ADDRBUF     $80    ; Address buffer for <Read memory> and
                             ; <Write memory> commands (MUST ALWAYS BE $0080)

; ST7 Product ICC I/O Definition ---------------------------------------------

PxDR    EQU   $08    ; I/O port register definition for ICC management
PxDDR   EQU   $09
  #DEFINE ICCCLK  0  ; Bit definitions for ICCCLK and ICCDATA pin allocation
  #DEFINE ICCDATA 1

; ST7 Product ICD Definition -------------------------------------------------
  #IF {ICD_Available}
DMCR    EQU   $60    ; ICD registers description if ICD available
DMCSR   EQU   $61    ; (refer to product datasheet for more details)
  #DEFINE MTR  6     ; MTR bit position in the DMCR
```

```
  #DEFINE RST  4     ; RST bit position in the DMCSR
  #ENDIF

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; ASSEMBLER SOFTWARE
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

  WORDS

  SEGMENT byte at DF00-DFDB 'ICC_monitor_seg' ; ICC System Memory for instance

;*******************************************************************************
  #IF {Not_Basic_ICC}
.ICC_send_byte         ; LOW LEVEL ICC PROTOCOL SUBROUTINE
;*******************************************************************************

; ICC SEND subroutine used by the ST7 to send a byte on ICCDATA

        ; before start and after end of subroutine:
        ;        ICCCLK = 1 as floating input
        ;        ICCDATA as floating input
        ; before start of subroutine:
        ;        Acc A = byte to send
        ; after end of subroutine:
        ;         X = 0
        ; ICCCLK levels: 0 for 17 Tcpu
        ;                1 for 43 Tcpu minimum

  LD    X,#$08                ; X = BitCounter
  BSET  PxDDR,#ICCDATA        ; Define ICCDATA as open-drain output

.icc_s1
  BSET  PxDR, #ICCDATA        ; ICCDATA=1 because tying line to 0 is faster
  RLC   A                     ; Copy bit to transmit into carry flag
  JRC   icc_s2                ; Test bit to transmit
  BRES  PxDR, #ICCDATA        ; If "0" set ICCDATA = 0
.icc_s2
  TNZ   A                     ; Dummy instructions to get a 2 x 3 Tcpu delay
  TNZ   A                     ; to insure the ICCDATA level
  BRES  PxDR ,#ICCCLK         ; Because BSET/BRES ICCDATA may set ICCCLK DR
  BSET  PxDDR,#ICCCLK         ; => ICCCLK output = 0
  TNZ   A                     ; Dummy instructions to get a 4 x 3 Tcpu delay
  TNZ   A                     ; for External Controller reaction time
  TNZ   A
  TNZ   A

  BRES  PxDDR,#ICCCLK         ; Release ICCCLK after 5 Tcpu

.icc_s3
  BTJF  PxDR ,#ICCCLK,icc_s3  ; Wait until ICCCLK = 1
                              ; Ext. Controller already detected 1 (TTL input)
  DEC   X
  JRNE  icc_s1                ; Check if the whole byte is sent

  BRES  PxDDR,#ICCDATA        ; If yes, define ICCDATA as floating input

        #IF {ICD_Available} ;++++++++++++++++++++++++++++++
  JRH   ICC_y_sent  ; Return when Y auto saving context +++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++

  RET

  #ENDIF  ; Preprocessing for ICC_send_byte routine
```

```
  ;++++++++++++++++++++++++++++++++++++++++++++++++

;*******************************************************************************
.ICC_monitor_reset    ; ICC reset vector destination
;*******************************************************************************

  ; ICC Communication * External Controller     = TTL INPUT / CMOS OUTPUT (HCT)
  ;                   * ST7 ICCCLK/ICCDATA pins = CMOS INPUT / CMOS OUTPUT
  ;                     I/O Configuration:
  ;                       ICCCLK  as floating input (std default reset config.)
  ;                       ICCDATA as floating input (std default reset config.)

  BTJF  PxDR, #ICCDATA, ICC_mode      ; To enter ICC Mode, ICCDATA must be kept
                                      ; low after reset
.ST_Internal_Use

  CLR   A
  CLR   X
  CLR   Y
  JP    RAMBEG

.ICC_mode
  LD    A,S                         ; To init the Stack for future <go> command
  SUB   A,#5
  LD    S,A

;*******************************************************************************
.ICC_monitor_trap      ; ICC trap vector destination
                       ; ICC Monitor MAIN ROUTINE
;*******************************************************************************

        #IF {ICD_Available} ;++++++++++++++++++++++++++++++
  BSET  DMCR, #MTR ; The ST7  is executing ICC Monitor ++++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++

; ---------------------------
; dummy bit synchronization:    ICCCLK tied to 0 for 16 Tcpu
; ---------------------------

.synchro1

  BTJF  PxDR ,#ICCCLK,synchro1 ; Wait until ICCCLK = 1
                               ; Ext. Controller already detected 1 (TTL input)
  BRES  PxDR ,#ICCCLK
  BSET  PxDDR,#ICCCLK          ; => ICCCLK output = 0

  MUL   X,A                    ; Dummy instruction to get a tempo to be sure
                               ; that Ext. Controller has detected 0 on ICCCLK
                               ; Clear Half Carry bit for auto-context saving

  BRES  PxDDR,#ICCCLK          ; Release ICCCLK
.synchro2
  BTJF  PxDR ,#ICCCLK,synchro2 ; Wait until ICCCLK = 1
                               ; Ext. Controller already detected 1 (TTL input)

  #IF {ICD_Available} ;++++++++++
  ; Automatic context saving  +++
  ;++++++++++++++++++++++++++++++

  BTJT  DMCSR,#RST,ICC_do_cmd ; No automatic saving context after reset

.ICC_autosave
  LD    A,#$0F
```

```
  ADD   A,#1                  ; Set half-carry flag
  LD    A,Y
  JRT   ICC_send_byte         ; Send Y reg. content to External Controller
.ICC_y_sent
  MUL   X,A                   ; Clear half-carry flag

  POP   A                     ; Two sequential POP to use only 5 byte stack
  POP   Y
  CALLR ICC_send_byte         ; Send CC reg. previous content
  LD    A, Y
  CALLR ICC_send_byte         ; Send Acc previous content

  LD    A,{ADDRBUF+1}
  CALLR ICC_send_byte         ; Send previous value of address buffer MSByte
  LD    A,ADDRBUF
  CALLR ICC_send_byte         ; Send previous value of address buffer LSByte

  LD    A,S
  CALLR ICC_send_byte         ; Send SP reg. previous value

  #ENDIF  ; Preprocessing for Automatic context saving
  ;++++++++++++++++++++++++++++++++++++++++++++++++++

; ----------------------------
; command interpreter
; ----------------------------

.ICC_do_cmd

  CALLR ICC_receive_byte      ; Receive command from External Controller

  RLC   A
  JRNC  ICC_cmd_write         ; cmd="0xxxxxx": <Write memory> command
  RLC   A
  JRNC  ICC_cmd_go            ; cmd="10xxxxx": <Go> or <Go New Context> cmd

        #IF {Not_Basic_ICC} ;++++++++++++++++++++++++++++++++++++++++++++++++++
  RLC   A                                                              ;+++
  JRNC  ICC_cmd_read          ; cmd="110xxxx": <Read memory> command    +++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  JRT   ICC_do_cmd            ; unknown command => Execute the next one

;*******************************************************************************
.ICC_receive_byte      ; LOW LEVEL ICC PROTOCOL SUBROUTINE
;*******************************************************************************

; ICC RECEIVE subroutine used by the ST7 to receive a byte from ICCDATA pin

        ; before start and after end of subroutine:
        ;        ICCCLK = 1 as floating input
        ;        ICCDATA as floating input
        ;        bit ICCCLK on Port x Data Register cleared
        ; after end of subroutine:
        ;        Acc A = byte received
        ;        X = 0
        ; ICCCLK levels: 0 for 28 Tcpu
        ;                (delay time from ICCCLK low to ICCDATA valid = 15Tcpu)
        ;                1 for 16 Tcpu minimum

  LD    X,#$08                ; X = BitCounter

.icc_r1
```

```
  BSET  PxDDR,#ICCCLK         ; => ICCCLK output = 0
  TNZ   A                     ; Dummy instructions to get a 5 x 3 Tcpu delay
  TNZ   A                     ; to ensure that Ext. Controller has set the data
  TNZ   A
  TNZ   A
  TNZ   A

  BTJF  PxDR,#ICCDATA,icc_r2  ; Update carry flag according to ICCDATA
.icc_r2
  RLC   A                     ; Save carry flag into Acc A

  BRES  PxDDR,#ICCCLK         ; Define ICCCLK as floating input

.icc_r3
  BTJF  PxDR ,#ICCCLK,icc_r3  ; Wait until ICCCLK = 1

  DEC   X
  JRNE  icc_r1                ; Check if the whole byte is received

        #IF {ICD_Available} ;++++++++++++++++++++++++++++++++++++++++++++++++++
  JRH   ICC_y_received     ; In this case, ICC_receive_byte is not a        +++
                           ; subroutine but a mere sequence of instructions +++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  RET

;*******************************************************************************
;  ICC COMMANDS LIBRARY
;*******************************************************************************

; ----------------------------------------------------
.ICC_cmd_write    ; <Write memory> command execution
; ----------------------------------------------------

        #IF {Not_Basic_ICC} ;++++++++++++++++++++++++++++++++++++++++++
  LD    Y, #$FF   ; Y reg is used as R/W cmd indicator: W=> Y=FFh   +++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++++++++++++++

.ICC_cmd_rw
  CALLR ICC_receive_byte      ; Receive MSB@ from External Controller
  LD    ADDRBUF,A             ; Store it into MSB@
  CALLR ICC_receive_byte      ; Receive LSB@ from External Controller
  LD    {ADDRBUF+1},A         ; Store it into LSB@
  CALLR ICC_receive_byte      ; Receive nb-1 of bytes to read or write
                              ; (this number must be in the range 0..128)
        #IF {Not_Basic_ICC} ;++++++++++++++++++++++++++++++++++++++++++++
  TNZ   Y                   ; Check if it is a read or write cmd       +++
  JREQ  ICC_cmd_read_start  ; R=> Y=00h / W=> Y=FFh                    +++
        #ENDIF ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  LD    Y,A                 ; Store into Y the number of bytes to write

.ICC_cmd_write_cont
  CALLR ICC_receive_byte      ; Receive data byte from External Controller
  LD    ([ADDRBUF.w],Y),A     ; Write data byte into the memory
  DEC   Y                     ; Next data byte
  JRPL  ICC_cmd_write_cont

.ICC_do_cmd2
  JRT   ICC_do_cmd            ; Execute next command

; ----------------------------------------------------
.ICC_cmd_go       ; <Go> command execution
```

```
; ------------------------------------------------

  LD    A,#DEVICE_ID.h        ; When exiting monitor after RESET: The device
  LD    {ADDRBUF},A           ; identifier is stored into the RAM at a fixed
  LD    A,#DEVICE_ID.l        ; address for all products (0080h/0081h).
  LD    {ADDRBUF+1},A         ; This information allows to have dev. tools
                              ; able to identify automatically the device.

  #IF {ICD_Available} ;++++++++++
  ; <Go new context> command  +++
  ;++++++++++++++++++++++++++++

  BTJT  DMCSR,#RST,ICC_go     ; Go new context if monitor not from RESET

.ICC_go_newctx
  CALLR ICC_receive_byte      ; Receive address buffer MSByte
                              ; from External Controller
  LD    {ADDRBUF+1},A         ; Store it at the right place in RAM
  CALLR ICC_receive_byte      ; Receive address buffer LSByte
  LD    ADDRBUF,A             ; Store it at the right place in RAM

  CALLR ICC_receive_byte      ; Receive new stack pointer (must be
  LD    S, A                  ; the same as the one send during
                              ; automatic context saving for normal operation)

  CALLR ICC_receive_byte      ; Receive Acc future content
  LD    Y, A

  CALLR ICC_receive_byte      ; Receive CC reg. future content
  PUSH  Y                     ; Two sequential PUSH to use only 5 byte stack
  PUSH  A

  LD    A,#1
  ADD   A,#$0F                ; Set half-carry flag
  JRT   ICC_receive_byte      ; Receive Y reg. future content
.ICC_y_received
  LD    Y, A

  #ENDIF  ; Preprocessing for <Go new context> command
  ;+++++++++++++++++++++++++++++++++++++++++++++++++

.ICC_go
  IRET                        ; Restore A, X, CC
                              ; Execute selected address or Continue

; ------------------------------------------------
  #IF {Not_Basic_ICC}
.ICC_cmd_read        ; <Read memory> command execution
; ------------------------------------------------

  CLR   Y                     ; Y reg is used as R/W cmd indicator: R=> Y=00h
  JRT   ICC_cmd_rw            ; in ICC_cmd_rw
.ICC_cmd_read_start
  LD    Y,A                   ; Store into Y the number of bytes to read

.ICC_cmd_read_cont
  LD    A,([ADDRBUF.w],Y)     ; Read data byte from memory
  CALL  ICC_send_byte         ; Send data byte to External Controller
  DEC   Y                     ; Next data byte
  JRPL  ICC_cmd_read_cont

  JRT   ICC_do_cmd2           ; Execute next command
```

```
  #ENDIF

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

    REPEAT                    ; Fill unused area with 00h
      BYTE    $00
val     CEQU  {$DFDC-*}
    UNTIL   {val eq $0}

;       *********************************
;       *  SYSTEM AND INTERRUPT VECTORS  *
;       *********************************

;******* ICD TRAP VECTOR ********************************

  SEGMENT byte at DFDC-DFDD 'icd trap vector'

    DC.W  ICC_monitor_trap       ; Used for ICD management

;******* ROMDUMP VECTORS ********************************
  SEGMENT byte at DFDE-DFDF 'romdump vector'

    DC.W  $FFFF                  ; ST Internal use only

;******* INTERRUPT VECTORS ******************************
  SEGMENT byte at DFE0-DFFF 'interrupt vectors'

    DC.W    {RAMBEG}             ; ITUSER13 - xxE0-xxE1h
    DC.W    {RAMBEG}             ; ITUSER12 - xxE2-xxE3h
    DC.W    {RAMBEG}             ; ITUSER11 - xxE4-xxE5h
    DC.W    {RAMBEG}             ; ITUSER10 - xxE6-xxE7h
    DC.W    {RAMBEG}             ; ITUSER9  - xxE8-xxE9h
    DC.W    {RAMBEG}             ; ITUSER8  - xxEA-xxEBh
    DC.W    {RAMBEG}             ; ITUSER7  - xxEC-xxEDh
    DC.W    {RAMBEG}             ; ITUSER6  - xxEE-xxEFh
    DC.W    {RAMBEG}             ; ITUSER5  - xxF0-xxF1h
    DC.W    {RAMBEG}             ; ITUSER4  - xxF2-xxF3h
    DC.W    {RAMBEG}             ; ITUSER3  - xxF4-xxF5h
    DC.W    {RAMBEG}             ; ITUSER2  - xxF6-xxF7h
    DC.W    {RAMBEG}             ; ITUSER1  - xxF8-xxF9h
    DC.W    {RAMBEG}             ; ITUSER0  - xxFA-xxFBh
    DC.W    {RAMBEG}             ; TRAP     - xxFC-xxFDh
    DC.W    ICC_monitor_reset    ; RESET    - xxFE-xxFFh

  END

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

APPENDIX 2 (ICC PC SOFTWARE FOR DOS AND WINDOWS 95/98)

Example of PC DOS software routines for driving the ICC board described in Section 8.
Note: These drivers cannot be used with Windows NT/2000/XP since these operating systems do not allow a user application to directly access the parallel port. Hence the calls to inportb(), outportb() in the software provided would not be accepted.

```
//*****************************************************************************
//      MODULE  icc_sw.c
//      AUTHOR  CMG_MCD Application Team
#define UPDATE  "19/11/01"
// DESCRIPTION: PC Parrallel Port interface for ST7 In-Circuit Commmunication
//              Compatible with ST7 Toolchain: programmer...
//*****************************************************************************


#include <stdio.h>
#include <pc.h>
#include <conio.h>
#include <dos.h>
#include <time.h>
#include "memtable.h" // Include file defining memtable (data to be downloaded)


//*****************************************************************************
// PC PARALLEL PORT & ICC INTERFACE DEFINITION
//*****************************************************************************


// PC Parallel Port Definition ..................................................
                        //   Pin  Name    Logic  I/O  Byte   Bit
                        // ---------------------------------------
#define PC_vpppulse 0   //    1   Strobe   -1     O   Ctl     0
#define PC_ack      0   //    2   D0        1     O   Data    0
#define PC_dataout  1   //    3   D1        1     O   Data    1
#define PC_init     2   //    4   D2        1     O   Data    2
                        //    5   D3        1     O   Data    3
#define PC_pulse    4   //    6   D4        1     O   Data    4
#define PC_vppon    5   //    7   D5        1     O   Data    5
#define PC_vpp12    6   //    8   D6        1     O   Data    6
                        //    9   D7        1     O   Data    7
                        //   10   Ack      -1     I   Stat    6
#define PC_datain   7   //   11   Busy     -1     I   Stat    7
                        //   12   Pe        1     I   Stat    5
                        //   13   Select    1     I   Stat    4
#define PC_reset    1   //   14   Auto feed-1     O   Ctl     1
#define PC_status   3   //   15   Error    -1     I   Stat    3
                        //   16   Int      +1     O   Ctl     2
                        //   17   Select   -1     O   Ctl     3
                        // 18..25 GND
                        // ------------------------------------------------------
                        //                      Register address
#define outdata(c) outportb(0x378,(c))  //    Input status register.....378h
#define outctrl(c) outportb(0x37A,(c))  //    Output control register...37Ah
#define instatus()  inportb(0x379)      //    Input status register.....379h
                        //  ------------------------------------------------------

#define PC_Datain   ( (instatus()>>PC_datain) & 1 )
#define PC_Status   ( (instatus()>>PC_status) & 1 )
```

```
//***************************************************************************
// BASIC PC PARALLEL PORT INTERFACE ROUTINES
//***************************************************************************

/* ---------------------------------------------------------------------------
ROUTINE NAME : SetD & ResD
INPUT/OUTPUT : Bit to be modified / None
DESCRIPTION  : Set or reset a data PC parallel port pins.
---------------------------------------------------------------------------*/
unsigned char data; // This global variable contains PC parallel port image.
void SetD (unsigned char bit) {
  data |= (1 << bit);
  outdata(data);
}
void ResD (unsigned char bit) {
  data &= ~(1 << bit);
  outdata(data);
}
/* ---------------------------------------------------------------------------
ROUTINE NAME : SetC & ResC
INPUT/OUTPUT : Bit to be modified / None
DESCRIPTION  : Set or reset a control PC parallel port pins.
---------------------------------------------------------------------------*/
unsigned char ctrl; // This global variable contains the control PC parallel
port image.

void SetC (unsigned char bit) {
  ctrl |= (1 << bit);
  outctrl(ctrl);
}
void ResC (unsigned char bit) {
  ctrl &= ~(1 << bit);
  outctrl(ctrl);
}
/* ---------------------------------------------------------------------------
ROUTINE NAME : VPP_Voltage
INPUT/OUTPUT : Input voltage (0V ,5V, 12V or other=pulses) / None
DESCRIPTION  : Set the VPP/ICCSEL pin configuration.
---------------------------------------------------------------------------*/
void VPP_Voltage (unsigned char voltage)
{
  switch (voltage) {
  case 12: /* Vpp=12V ------------------------------*/
    SetD(PC_vppon);          /* and connect to VPP */
    SetC(PC_vpppulse);
    SetD(PC_vpp12);      /* 12V output from ST662A */
    break;
  case 5: /* Vpp=5V -------------------------------*/
    ResD(PC_vpp12);       /* 5V output from ST662A */
    SetC(PC_vpppulse);
    SetD(PC_vppon);        /* and connect from VPP */
    break;
  case 0: /* Vpp=0V -------------------------------*/
    ResD(PC_vpp12);       /* 5V output from ST662A */
    ResD(PC_vppon);     /* and disconnect from VPP */
    break;
  default: /* Vpp pulse for Fixed 256Tcpu Window --*/
    ResC(PC_vpp12);       /* 5V output from ST662A */
    ResC(PC_vpppulse);   /* when RESET rising edge */
```

```
    SetD(PC_vppon); /* and pulse generation on VPP */
    break;
  }
}
//***************************************************************************
// ICC COMMUNICATION ROUTINES
//***************************************************************************

/* ----------------------------------------------------------------------------
ROUTINE NAME : ICC_send
INPUT/OUTPUT : Byte to be sent / None
DESCRIPTION  : ICC send routine via PC parallel port.
----------------------------------------------------------------------------*/
void ICC_send (unsigned char byte) {

  unsigned char i;

  for (i=0; i<8; i++)  {   // Send bit by bit the byte to be sent
    if (byte & 0x80)
       SetD(PC_dataout);   // Release ICCDATA line (data bit to transmit)
    else
       ResD(PC_dataout);   // Force ICCDATA line to 0 (data bit to transmit)
    byte = byte << 1;      // Next bit to be send is the MSB
    SetD(PC_ack);          // To release ICCCLK to for next communication bit
    ResD(PC_ack);
    while (!PC_Status) {   // Wait for PC_status line high (meaning that ST7 as
      if (kbhit())         // released ICCCLK to 1 and then force it back to 0)
        exit(1);           // If key pressed then exit
    }
  }
  SetD(PC_dataout);        // Release ICCDATA line
}

/* ----------------------------------------------------------------------------
ROUTINE NAME : ICC_receive
INPUT/OUTPUT : None / Received byte
DESCRIPTION  : ICC receive routine via PC parallel port.
----------------------------------------------------------------------------*/
unsigned char ICC_receive (void) {

  unsigned char i, byte = 0;

  for (i=0; i<8; i++)  {   // Receive bit by bit the byte to be received
    SetD(PC_ack);          // To release ICCCLK to for next communication bit
    ResD(PC_ack);
    byte = byte << 1;      // Prepare to receive next bit in the buffer
    while (!PC_Status) {   // Wait for PC_status line high (meaning that ST7 as
      if (kbhit())         // released ICCCLK to 1 and then force it back to 0)
        exit(1);           // Data is then available on ICCDATA (and PC_datain)
    }                      // If key pressed then exit
    if (PC_Datain) byte |= 0x01;   //    > Received bit is a 1
    else byte &= 0xFE;             //    > Received bit is a 0
  }
  return (byte);           // Return received byte
}

//***************************************************************************
// ICC MODE ENTRY ROUTINES
//***************************************************************************

/* ----------------------------------------------------------------------------
ROUTINE NAME : ICC_Enter_CtrlWin
INPUT/OUTPUT : Nb of pulse to be applied / None
```

```
DESCRIPTION  : ICC Mode entry with Controlled Window.
-------------------------------------------------------------------------------*/
void ICC_Enter_CtrlWin (unsigned char pulse) {

  unsigned char i;

  VPP_Voltage(0);   // To reset pulse counter, when ICCSEL is used.
  ResC(PC_reset);   // RESET=5V: To get a low level pulse afterward
  ResD(PC_dataout); // Force ICCCLK to 0 and then initialized DATA latch
  SetD(PC_init);    // to avoid conflict on ICCDATA during pulses

  SetC(PC_reset);   // RESET=0V: To start a ICC RESET phase
  SetD(PC_ack);  // When ICCCLK is used to reset counter, to release ICCCLK
  ResD(PC_ack);  // (=1) and then perform counter reset when RESET becomes 0.
  delay(1); // To be sure that the RESET is taken into account (RESET filter)

  // Open the pulse window
  VPP_Voltage(5); // Open pulse window, when ICCSEL is used for counter reset
  ResD(PC_init);  // Force ICCCLK to 0 and then open the pulse window,
  SetD(PC_init);  // when ICCCLK is used for counter reset.

  ResD(PC_pulse);            // Generate N pulses on PC_pulse (PC_pulsen=0
  for (i=0; i<pulse; i++) {     // after PC parallel port initialization)
    SetD(PC_dataout);
    ResD(PC_dataout);
  }
  // Close the pulse window
  ResC(PC_reset); // RESET=5V: To close the pulse window and then start the
                  // ST7 ICC program execution (ICC_Data=0 thanks to PC_pulse)
  SetD(PC_ack);   // When ICCCLK is used for counter reset, to release ICCCLK
  ResD(PC_ack);   // (=1) to be ready for the ICC synchro bit

  while (!PC_Status) {   // Wait for PC_status line high (meaning that ST7 as
    if (kbhit())         // released ICCCLK to 1 and then force it back to 0)
      exit(1);           // If key pressed then exit
  }

  SetD(PC_pulse);        // Disable PC_pulse capability on ICCDATA
}

/* ------------------------------------------------------------------------------
ROUTINE NAME : ICC_Enter_FixedWin
INPUT/OUTPUT : None / None
DESCRIPTION  : ICC Mode entry with Fixed 256Tcpu Window.
-------------------------------------------------------------------------------*/
void ICC_Enter_FixedWin (void) {

  VPP_Voltage(0);   // To reset pulse counter, when ICCSEL is used.
  ResC(PC_reset);   // RESET=5V: To get a low level pulse afterward
  ResD(PC_dataout); // Force ICCCLK to 0 and then initialized DATA latch
  SetD(PC_init);    // to avoid conflict on ICCDATA during pulses

  SetC(PC_reset);   // RESET=0V: To start a ICC RESET phase
  SetD(PC_ack);  // When ICCCLK is used to reset counter, to release ICCCLK
  ResD(PC_ack);  // (=1) and then perform counter reset when RESET becomes 0.
  delay(1); // To be sure that the RESET is taken into account (RESET filter)

  VPP_Voltage(0xFF);  // Enable VPP pulses

  ResD(PC_init);    // To initialize ICCCLK to ground to make the ST7 wait the
  SetD(PC_init);    //  Controller after fetching RESET vector

  ResD(PC_pulse);   // ICCDATA forced to ground to make the ST7 enter ICC Mode
```

```
  ResC(PC_reset); // RESET=5V: Raises RESET to start pulse sequence and then
                  // start the ST7 ICC program execution (ICC_Data=0 thanks to
                  // PC_pulse)
  SetD(PC_ack);   // To release ICCCLK (=1) to be ready for the ICC synchro bit
  ResD(PC_ack);

  while (!PC_Status) {   // Wait for PC_status line high (meaning that ST7 as
    if (kbhit())         // released ICCCLK to 1 and then force it back to 0)
      exit(1);           // If key pressed then exit
  }

  SetD(PC_pulse);        // Disable PC_pulse capability on ICCDATA
}


//***************************************************************************
// HIGH LEVEL INTERFACE ROUTINE
//***************************************************************************

/* ------------------------------------------------------------------------
ROUTINE NAME : ICC_Cmd_Write
INPUT/OUTPUT : None / None
DESCRIPTION  : ICC write command routine.
------------------------------------------------------------------------*/
void ICC_Cmd_Write (void) {

  unsigned int nbbyte, address, data;

  printf("  >>> Write Command\n");
  printf("        [@MSB:@LSB] 0x:"); scanf("%X",&address);
  printf("        [Byte nb]   0x:"); scanf("%X",&nbbyte);
  if (nbbyte==0 || nbbyte>0x81) return; // Check byte number limits [1..129]

  ICC_send(0x00);                              // ICC write command
  ICC_send((unsigned char) (address>>8));      // - Param: address MSB
  ICC_send((unsigned char) address);           // - Param: address LSB
  ICC_send((unsigned char) (nbbyte-1));        // - Param: byte number (N-1)

  for (; nbbyte>0; nbbyte--) {  // Get data bytes in Hex and send it to ST7
    printf("        [@=0x%4X]  0x:",address+nbbyte-1); scanf("%X",&data);
    ICC_send((unsigned char) data);
  }
  ICC_send(0xFF);    // Dummy command to make the ST7 execute the last write.
}

/* ------------------------------------------------------------------------
ROUTINE NAME : ICC_Cmd_Read
INPUT/OUTPUT : None / None
DESCRIPTION  : ICC read command routine.
------------------------------------------------------------------------*/
void ICC_Cmd_Read (void) {

  unsigned int nbbyte, address, data;

  printf("  >>> Read Command\n");
  printf("        [@MSB:@LSB] 0x:"); scanf("%X",&address);
  printf("        [Byte nb]   0x:"); scanf("%X",&nbbyte);
  if (nbbyte==0 || nbbyte>0x81) return; // Check byte number limits [1..129]

  ICC_send(0xC0);                              // ICC read command
  ICC_send((unsigned char) (address>>8));      // - Param: address MSB
  ICC_send((unsigned char) address);           // - Param: address LSB
```

```
  ICC_send((unsigned char) (nbbyte-1));      // - Param: byte number (N-1)

  for (; nbbyte>0; nbbyte--) {               // Receive from ST7 data bytes
    printf("        [@=0x%4X]  0x%2X",address+nbbyte-1,ICC_receive());
  }
  ICC_send(0xFF);    // Dummy command to make the ST7 execute the last read.
}

/* ---------------------------------------------------------------------------
ROUTINE NAME : ICC_Cmd_Go
INPUT/OUTPUT : None / None
DESCRIPTION  : ICC go command routine.
---------------------------------------------------------------------------*/
void ICC_Cmd_Go (void) {
  printf("\n  >>> GO and Exit ICC Program");
  ICC_send(0x80);  // ICC go command
  SetD(PC_ack);    // To release ICCCLK to make the ST7 execute the command.
  ResD(PC_ack);
}

/* ---------------------------------------------------------------------------
ROUTINE NAME : ICC_Cmd_GoNewContext
INPUT/OUTPUT : None / None
DESCRIPTION  : ICC go new context command routine.
---------------------------------------------------------------------------*/
void ICC_Cmd_GoNewContext (void) {
  unsigned int context;

  printf("\n  >>> GO NEW CONTEXT and Exit ICC Program\n");
  ICC_send(0x80);  // ICC go command
  printf("         [@0x0081] 0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  printf("         [@0x0080] 0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  printf("         [ S ]     0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  printf("         [ A ]     0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  printf("         [ CC ]    0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  printf("         [ Y ]     0x:"); scanf("%X",&context);
                                    ICC_send((unsigned char) context);
  SetD(PC_ack);    // To release ICCCLK to make the ST7 execute the command.
  ResD(PC_ack);
}

/* ---------------------------------------------------------------------------
ROUTINE NAME : ICD_WaitForAbortBreak
INPUT/OUTPUT : None / None
DESCRIPTION  : Wait for BREAKPOINT or ABORT (ICD driver routine).
---------------------------------------------------------------------------*/
void ICD_WaitForAbortBreak (void) {

  // Wait for BREAKPOINT or ABORT.............................................
  printf("  >>> APPLICATION RUNNING... Press a key for ICD ABORT \n");
  while (!PC_Status) {   // Wait for PC_status line high (meaning that ST7 as
    if (kbhit()) {       // released ICCCLK to 1 and then force it back to 0)
      getch();           // To free the keyboard buffer
      ResD(PC_dataout);  // If key pressed then ABORT is generated (ICCDATA=0)
      ResD(PC_pulse);    // Enable PC_pulse capability on ICCDATA
  } }

  // Check which ICD event occured...........................................
```

```
  printf("  >>> STOPPED by ICD ");
  if (PC_Datain) printf("BREAKPOINT\n");
  else           printf("ABORT\n");
  SetD(PC_dataout);      // Release ICCDATA line
  SetD(PC_pulse);        // Disable PC_pulse capability on ICCDATA

  // AUTOMATIC CONTEXT SAVING.....................................................
  printf("  >>> AUTOMATIC CONTEXT SAVING\n");
  printf("       [ Y ]     0x:%2X\n", ICC_receive());
  printf("       [ CC ]    0x:%2X\n", ICC_receive());
  printf("       [ A ]     0x:%2X\n", ICC_receive());
  printf("       [ 0x81 ]  0x:%2X\n", ICC_receive());
  printf("       [ 0x80 ]  0x:%2X\n", ICC_receive());
  printf("       [ S ]     0x:%2X\n", ICC_receive());
}
```

SUMMARY OF CHANGES

Date	Rev	Description
Sept-2001	1.0	"ICC Protocol" reference manual document extract from "ICC & Flash Programming" reference manual from Feb-2001.
Mar-2002	1.1	Page 6 Add a short definition of the ST7 RESET phase in the GLOSSARY. Page 8 ST72FLite00 product added. Note 4: DM protection capability information added for specific devices. Page 17 New section to describe how to switch from ICC receive to ICC send. Page 18 Add a short definition of the ST7 RESET phase. Page 20 Add device identifier and System Memory version nibble definition. Page 27 System memory addresses added for ST72FLite00 and ST72F344. Page 21 In Figure 12, pin 10 of HE10 connector is changed from NC to GND. Note 1 modified for ST programming tools. Note 2 modified: remove OSC2 to be forced to GND. Page 29 Schematic modified: - ic2 pin 12 connected to PC_data_out instead of Vss. - 74HCT00 replaced by 74ACT00. - ICC10 connector added to GND. - C11 removed. Page 31 Schematic modified: - STT3PF30L transistor reference added. - 74HCT00 replaced by 74ACT00. - r6 resistor removed. - J1 jumper with dedicated note added to protect XFlash and ROM against 12V on ICCSEL/VPP pin. Page 32 Timing diagram modified to fit new schematics. Page 33 Schematic modified: - new counter management part. - r6 resistor removed. - J1 jumper with dedicated note added to protect XFlash and ROM against 12V on ICCSEL/VPP pin. Page 35 Reset connectivity chapter added. Page 36/Page 37Timing diagram modified to fit new schematics (page 27 & 30). Note added. Page 40 Add new timing for the validation of the ABORT signal. Page 59 New software example in Appendix 2 to be coherent with schematic modifications. Page 66 In Appendix 3, add Multi DM feature. Add a reference to table 1 for DM availability in ST7 products. In figure, add WP bit, slave interface and note. Page 67 MTR bit description: add DM register write protection when MTR=0. Page 69 STE and STF bit locations corrected. WP bit description added. Page 69 BRW bit clear sequence description modified. Page 72 MTR and WP protection description added. Breakpoint functional description updated. Page 73 New DM timing description chapter added. Page 77 Multi DM chapter added. Page 84 Summary of changes chapter added.
Aug- 2002	1.2	Page 6 Note 6: new information concerning the default option value with external clock ICC mode. Page 8 CAUTION added for a safe ICCCLK configuration during reset.
Jan- 2003	1.3	Page 6 Replaced ST7FLite00 by ST7FSuperlite. Removed Debug Module (DM) from ST72F561 and ST72F264. Page 23 Replaced ST7FLite00 by ST7FSuperlite.
Apr- 2003	1.4	Page 8 Added ST7SCR and ST7FSCR to table. Page 10 Add heading Special case: Devices without ICCSEL pin Page 12 Added note on devices without RESET pin. Page 79 Added appendix 4

Date	Rev	Description
Nov- 2003	1.5	Page 8 Added ST7LITES2, ST7LITES5, ST7DALI and ST7 MC. Removed ST7Superlite and ST72344 Modified note 10 Added notes 12, 13 and 14 below section Table 1. on page 8 Page 11 Modified caution in section 3.1 on page 10 page 14 Added Section 3.3 External Clock/Application Clock ICC modes Page 27 Added ICC addresses for ST72264 ROM. Page 59 Added note about Win NT/2K/XP Page 82 Added appendix 5
May-2005	2	Revision number incremented from 1.5 to 2 due to Internal Document Management System change Removed device-specific tables to move into the "Flash Programming Quick Reference Manual" Removed any references to Debug Module to place into the “Debug Module Reference Manual” Page 30 Added a reference to STICK manufacturing files Page 31 Added Note on ST662A input voltage
Feb-2006	3	Page 13 Added paragraph on reserved area protection in Section 3.3