Microcontrollers

ApNote AP166201

☑ or ☐ additional file
AP166201.EXE available

Connecting the SSC (synchronous serial interface) of Infineon C16x microcontrollers to a time-division-multiplex (TDM) interface with 32 8-bit timeslots and 8 kHz frames (PCM highway).

Connecting the SSC of C16x to a TDM interface at 2.048 Mbit/s (PCM highway)

In digital transmission systems for telecommunications multiplex equipment operating at 2048 kbit/s is very common. In a lot of applications it is very useful for a control unit (microcontroller) to have access to information from or to that PCM time-division-multiplex equipment. This access is done by a synchronous full duplex serial interface which allows communication between different devices. The recommendations for this interface are given in ITU G.703, G.704 and G.732.

Author: Wolfgang Boelderl-Ermel, COM WN SE

1999-07, Rel. 01

AP166201 ApNote - Revision History		
Actual Revision : Rel.01	Previous Revision: none	
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release)

Edition 1999-07

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

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

Information

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

Warnings

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

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

0 List of abbreviations

CO	Central office
COT	Central office terminal
DCLK	Data clock
DD	Data downstream
DU	Data upstream
EX2IN	Fast external interrupt 2 input of C16x
FSC	Frame synchronisation clock
ITU	International Telecommunication Union
MDSL	Medium bitrate digital subscriber line
MRST	SSC master receive, slave transmit input/output of C16x
MTSR	SSC master transmit, slave receive output/input of C16x
PEC	Peripheral event controller of C16x
PCM	Pulse code modulation
RT	Remote terminal
SCLK	SSC master / slave clock output/input of C16x
SSC	Synchronous serial interface of C16x

1 Introduction

Abstract

In digital transmission systems for telecommunications multiplex equipment operating at 2.048 Mbit/s is very common. In a lot of applications it is very useful for a control unit (microcontroller) to have access to information from or to that PCM (pulse code modulation) time-division-multiplex equipment, which is often called PCM highway. This access is done by a synchronous full duplex serial interface which allows communication between different devices. The recommendations for this interface are given in ITU G.703, G.704 and G.732.

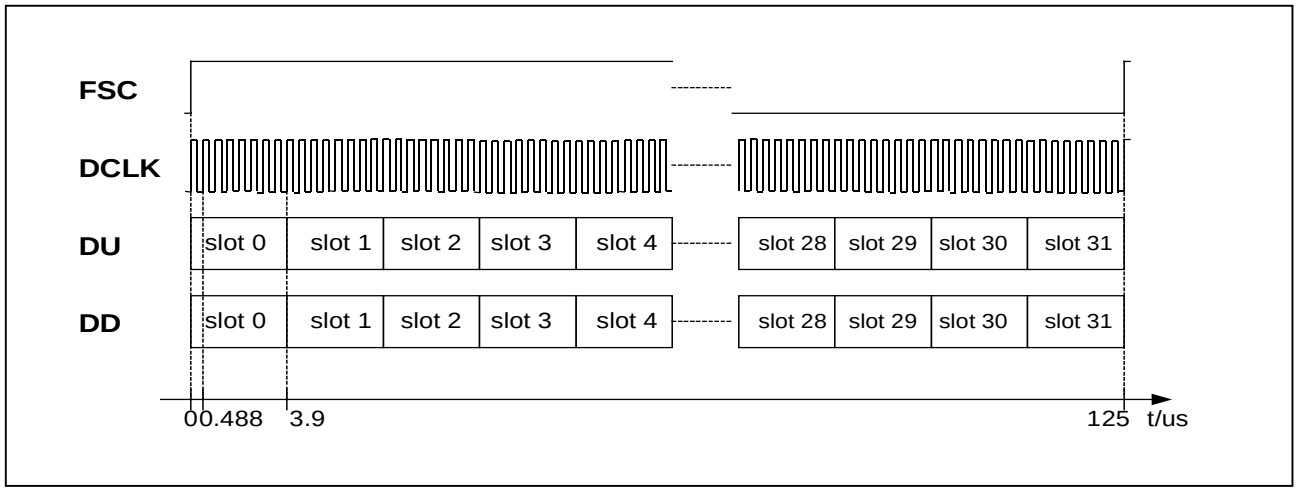

Figure 1 Timing of PCM highway (time-division-multiplex at 2.048 MHz DCLK)

At this interface with a rising edge at FSC (frame synchronisation) the start of a frame is signalled. This frame consists of 32 time slots (slot 0 .. 31), each time slot with 8 bits. So, a certain slot has to wait for 31 time slots until it may transfer data again. This results in a total bit rate of 64 kbit / s (= 2.048 Mbit /s : 32) per slot. The data transfer is clocked by DCLK (2.048 MHz). DU (data upstream) serves as data line from a device like a telephone to the central office. DD (data downstream) is used for data transfer vice versa. DU and DD are push/pull pins if active and open drain / tristate drivers if passive.

The 16-bit microcontroller family C16x is a good choice as control unit fitting to PCM time-division-multiplex equipment. The fast synchronous serial interface (SSC) together with an architecture supporting real time critical applications makes this microcontroller family ideal for PCM time-division-multiplex equipment, like at PCM highway (32 8-bit timeslots and 8 kHz frame).

This brings a reduction of system costs and gives more flexibility in the system design. A lot of applications may arise out of SSC´s capability to fit to PCM highway.

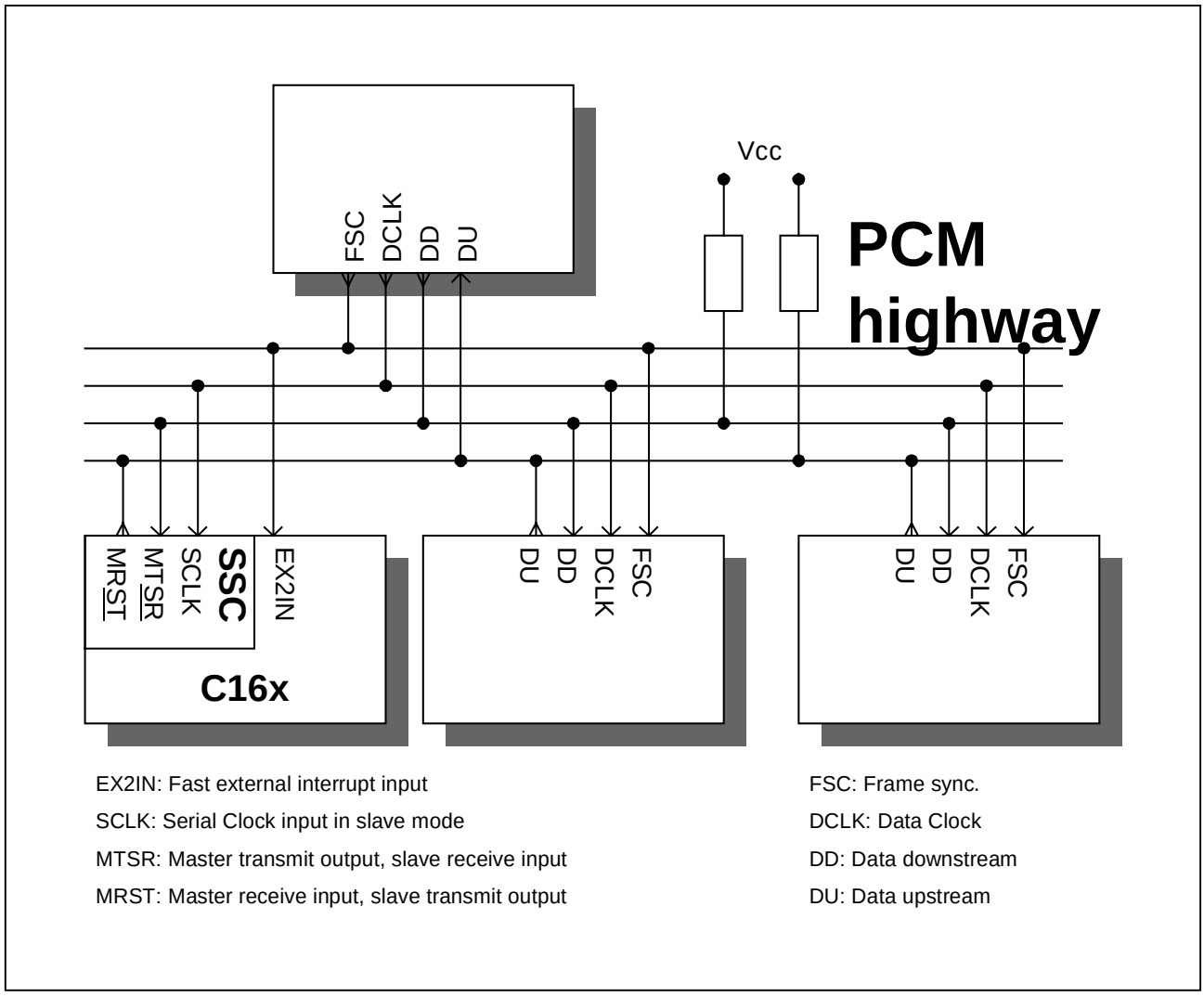

Figure 2 System configuration with PCM highway

This application note describes the software implementation for initialising and running the microcontroller C16x with a PCM time-division-multiplex system. When talking about C16x not a dedicated microcontroller is meant, but all members of that family like C161V, C161K, C161O, C161OR, C161PI. These microcontrollers are very convenient for telecommunication applications.

The basic software drivers are embedded into a testshell which allows to demonstrate the communication between several devices. Finally, one chapter gives information about downloading that demonstration code to an evaluation board and connecting it to a PCM time-division-multiplex system. One chapter dealing with benchmarks for the performance of the solution will help design engineers to have an early estimation about the possibilities of this solution.

2 Example for the access to PCM time-division-multiplex equipment

System Description

A PCM-x system that is also known under the name Pairgain systems or Digital Added Main Line (DAML) is a system that concentrates several analog telephone lines into one digital line. It is used between an analog central office (CO) and an analog subscriber telephone. All voice data is digitised and transmitted via the digital line (DSL). In addition signalling information is transmitted to indicate ringing, outgoing-call, TTX-pulses, and so on. The part of the system closer to the CO is called Central Office Terminal (COT) and the part closer to the subscriber is called remote Terminal (RT).

This kind of application is used to use a given amount of already installed telephone lines with a higher efficiency. Instead of one analog phone call with a PCM-x system several analog phone calls can be transmitted at the same time. A typical number of concentrated lines is between 2 and 11.

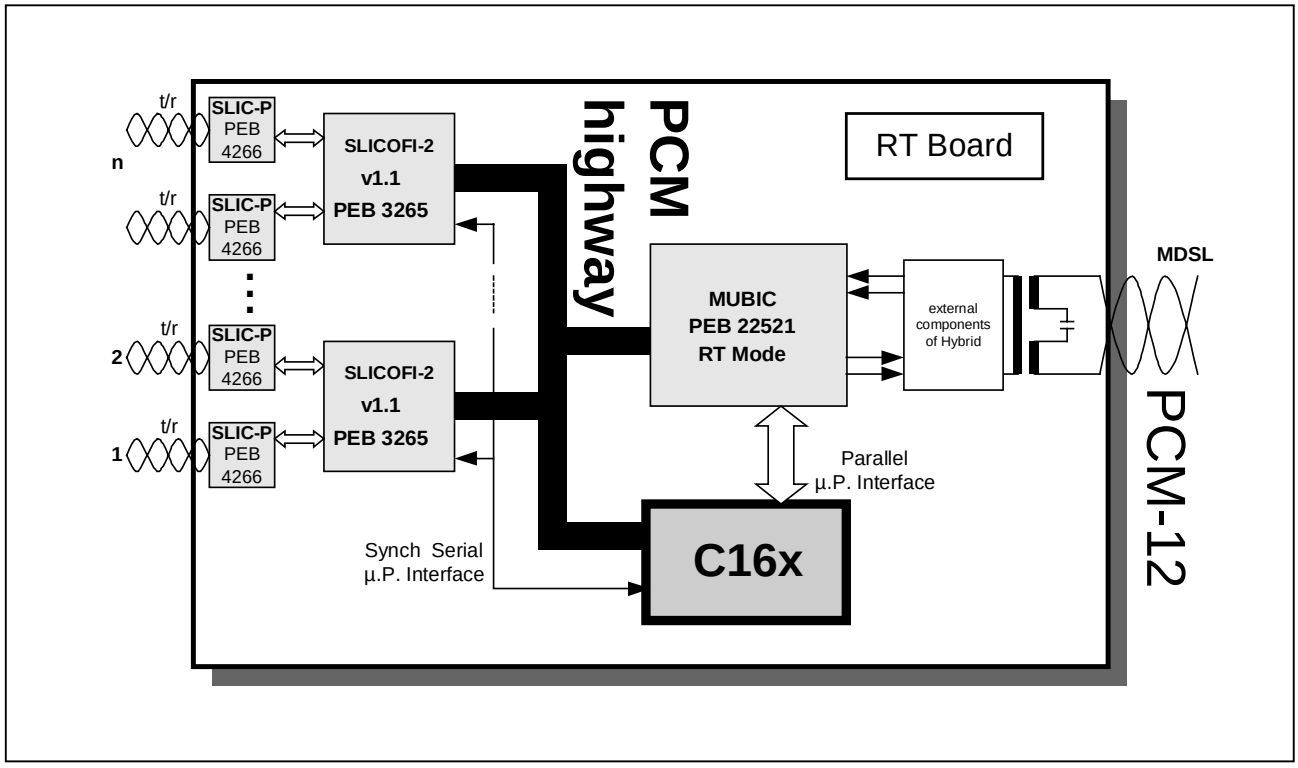

Figure 3 Example of PCM highway in MDSL application

Hint: There may be confusion about using the expression PCM-x when we talk about e.g. PCM-12 and PCM time-division-multiplex systems. PCM-12 means the transmission system via the telephone lines, in the example above solved by MDSL. On the other hand,

PCM highwa means the on-board (here: remote terminal RT board) connection of several devices.

On this board a solution has to be found which lets the microcontroller C16x communicate to the on-board PCM highway system. This solution makes use of the synchronous serial interface (SSC), which is available on each C16x chip.

By using the strong real time capability of the C16x architecture (PEC transfers) the SSC interface may be configured to work as a PCM time-division-multiplex interface.

3 Solution

Synchronisation

The process of synchronisation between PCM highway and SSC is triggered by a FSC-rising edge (start of frame). The CPU gets aware of this rising edge by connecting the signal FSC to one of the fast external interrupt pins. After detecting one rising edge (external interrupt) the CPU has to switch on and initialise the C16x synchronous serial interface (SSC).

The C16x synchronous serial interface runs in slave mode. So, it expects an external clock signal for its shift register. This will be the bus driven clock line DCLK. In addition the pin MRST (DU) will work as an open drain transmitter and MTSR (DD) as the receive pin of the C16x.

There is a certain delay between the action of detecting FSC-rising edge and switching on the SSC. Therefore there will be a certain slippage when the SSC starts to recognise the first rising edge at DCLK and the first bit (details see at chapter 7). This slippage has to be taken into account when a certain PCM slot has to be decoded. The slippage may be verified in the next step. Important: this slippage causes no loss of data, because the SSC is able to send / receive a continuous datastream. The effects of slippage can easily be compensated when the slippage is always constant. This is discussed in detail in chapter 7. recognize

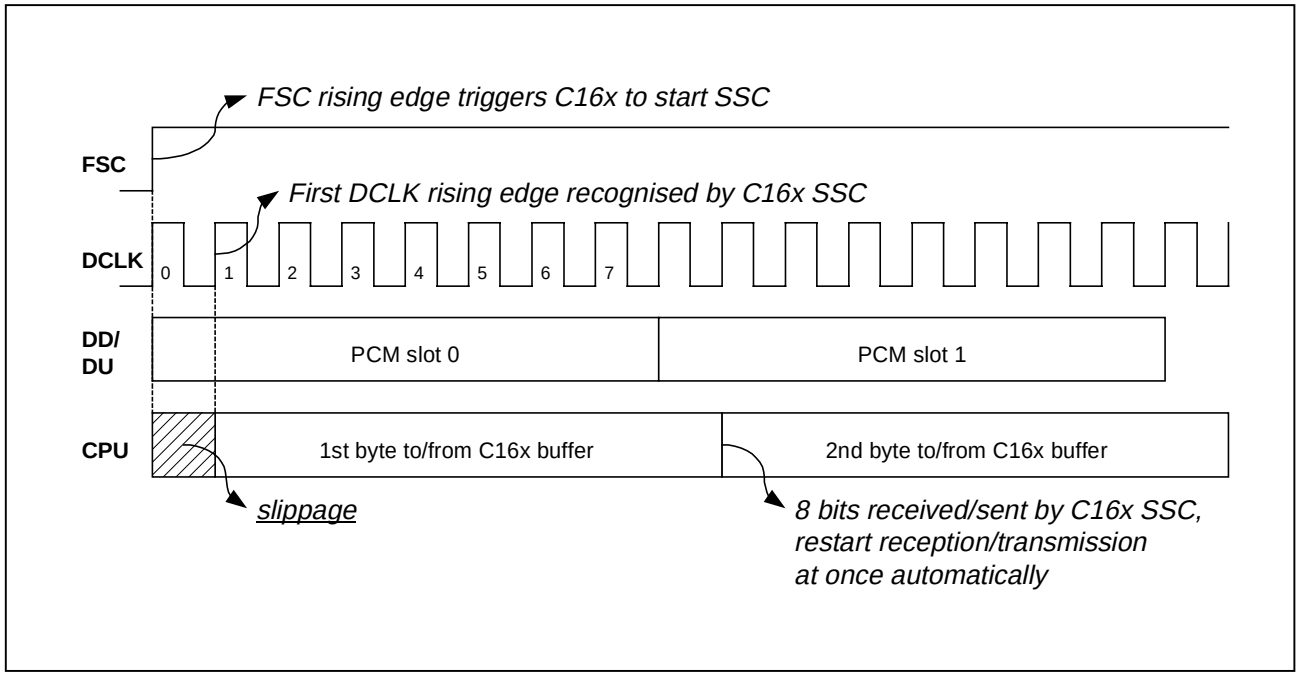

Figure 4 Synchronisation: Slippage

Verification

After the process of synchronisation the process of verification may take place. Now it is checked whether the synchronisation was successful. For this procedure we have to differ between two kinds of systems.

Verification of a system with feedback

If there is a system where at a certain point of time a well known data can be received by the C16x SSC (from PCM time-division-multiplex system), the C16x has the chance to calculate the slippage out of the received data of that PCM slot. (C16x receives the continuos bitstream. There is no loss of data.)

This may be done by a bus master who sends for example the bitstream 01101110 in PCM slot 0 to the C16x. Usually, with one bit slippage the SSC will receive the bitstream 1101110X (X means the first bit of the next slot). Of course the test may be possible vice versa: C16x sends well defined data in a certain PCM timeslot. Now, it must get a feedback from another device, if the slot was correctly transmitted.

This application note includes a software package which is called *PCMuCl*. If a system with feedback is designed this software gives a complete example. Detail see in chapter 8 and 9.

Verification of a system without feedback

In a lot of applications a system with feedback is not possible because the PCM slots contain for example speech data from a codec. This data is completely random and so there is no chance for C16x to verify the synchronisation by an external partner.

In this case, C16x has to check the synchronisation by itself. This is done by means of the C16x timer T5. (This timer will not be allocated for the whole time but only for this short period of verification. After this measurement T5 resources are completely free again for other functions of the application.)

In a system without feedback, the next (after synchronisation) FSC-rising edge (start of frame) triggers the start of timer T5. The SSC-receive interrupt (after one slot was received) stops timer T5.

With these two events triggering T5 the relationship between FSC (PCM) and the SSC (CPU) may be calculated.

This application note includes a software package which is called *PCMuC* and a package with the name *PCMuCo*. If a system without feedback is designed this software gives a complete example. Detail see in chapter 8 and 9. *PCMuC* and *PCMuCo* are nearlly the same. *PCMuCo* is a slightly speed optimised version.

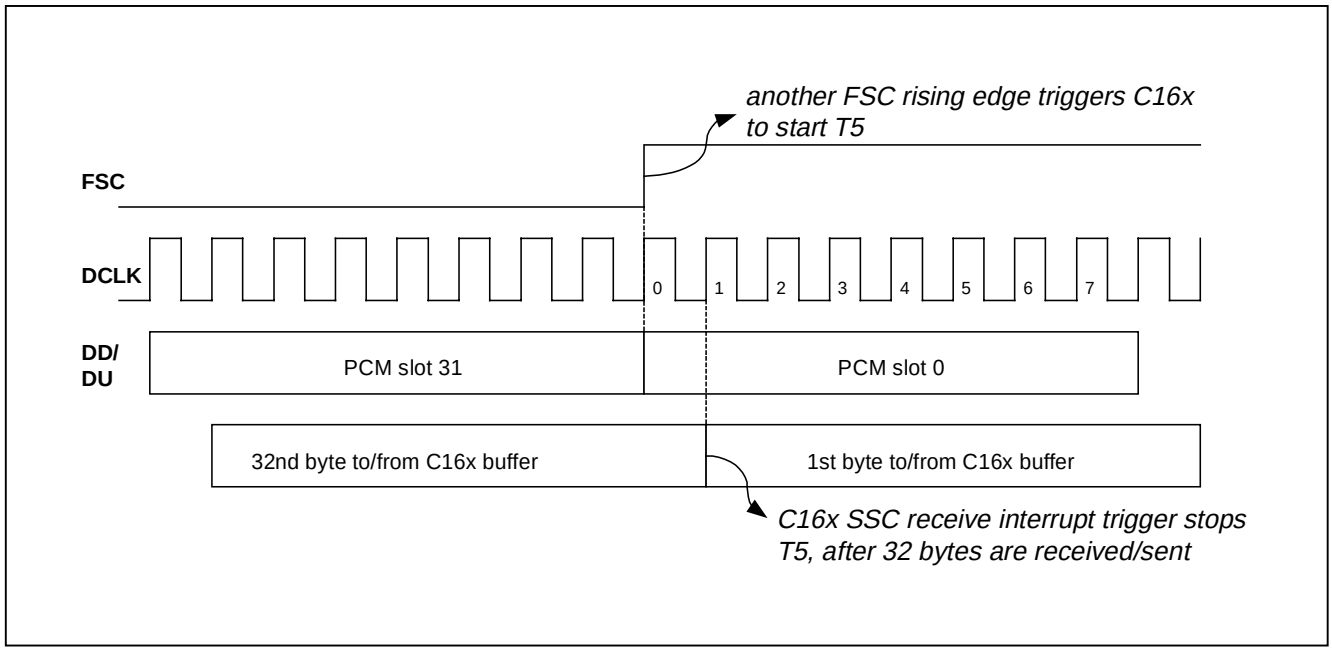

Figure 5 Verification: Slippage

It should be noticed that the timers of GPT2 (general purpose timer unit 2) should be used, because they have a higher resolution than those of GPT1. With a resolution of 250 ns at a C16x CPU frequency of 16 MHz (direct drive) a bitstream of 2.048 Mbit / s (488 ns / bit) will result in a count value of 1, if there is one bit slippage.

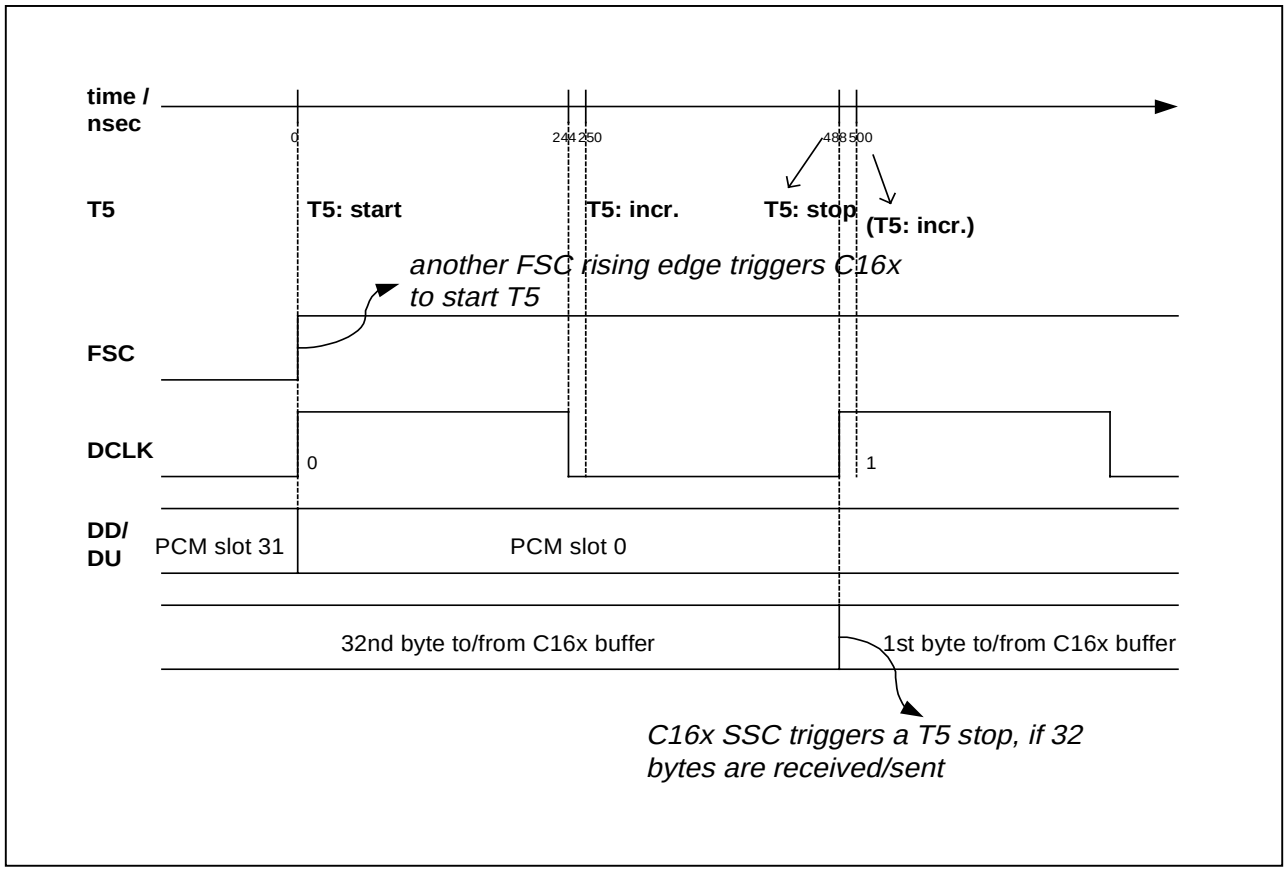

Figure 6 Details of verification

If the expected value is not meet the process of synchronisation should be repeated.

Finally the C16x is ready to access each slot of the PCM highway in receive and transmit direction.

PEC for synchronisation

A closer look at the implementation of the access to PCM highway by means of the SSC on the C16x shows that the PEC channels are an extremely useful and efficient feature of the C16x architecture.

In the process of synchronisation the detection of the FSC rising edge is done by a fast external interrupt. At the kitCON161 starter kit solution this is done by pin EX2IN. This pin is configured to detect a rising edge and to generate an interrupt request in the C16x. This interrupt request is linked to a PEC channel (here: PEC4). This link means that if the interrupt request occurs there will be not the traditional way of serving it by an interrupt handler with the overhead of saving the system state before the first useful instruction is executed and the overhead of restoring system state after the last useful instruction is

executed. (e.g. 5 instruction cycles at 16 MHz CPU clock rate are 625 ns. This is more than one bit at a 2 Mbit / s datastream.)

In this case instead of the interrupt handler a transfer by the PEC channel is executed. For the PEC channel a source register and a destination register was loaded during chip initialisation. Now, triggered by the event of FSC-rising edge a PEC transfer moves the contents of the address in the source register to that address which is specified in the destination register. In our case the source register contains the address of a static variable where the initialisation value of the SSC is stored. This value will let the SSC start if the destination register contains the address of the register SSCCON. This PEC transfer just “steals” one instruction cycle from the CPU. So, with a minimum of delay the SSC is switched on after a FSC-rising edge.

PEC for verification and running system

The same technique is used during verification when timer T5 is started and stopped. And finally this technique is used to transfer 8 received bits to the C16x buffer memory (source register contains address of SSC receive buffer, destination register a correct address in C16x RAM buffer). Of course the transmission is handled in the same way (source register contains a correct address in C16x RAM buffer, destination register the address of SSC transmit buffer). So, these PEC transfers allocate only a minimum of CPU performance compared to a transfer driven by interrupt handlers.

Basically, two buffers would be enough to handle the incoming bitstream and the outgoing bitstream. In this application note 2 buffers for each direction are used (this results in 4 buffers, 32 bytes (or 16 words) each). So, if the PEC transfer works on one buffer in transmit direction (e.g. uC_to_PCM_even), the user application on the C16x may work on the other buffer (uC_to_PCM_odd) which will be the data source for the following transmission. This technique avoids a collision between reading out and writing to the buffers. The same situation applies to the receive direction. The access is controlled by the bit variable *even_or_odd*.

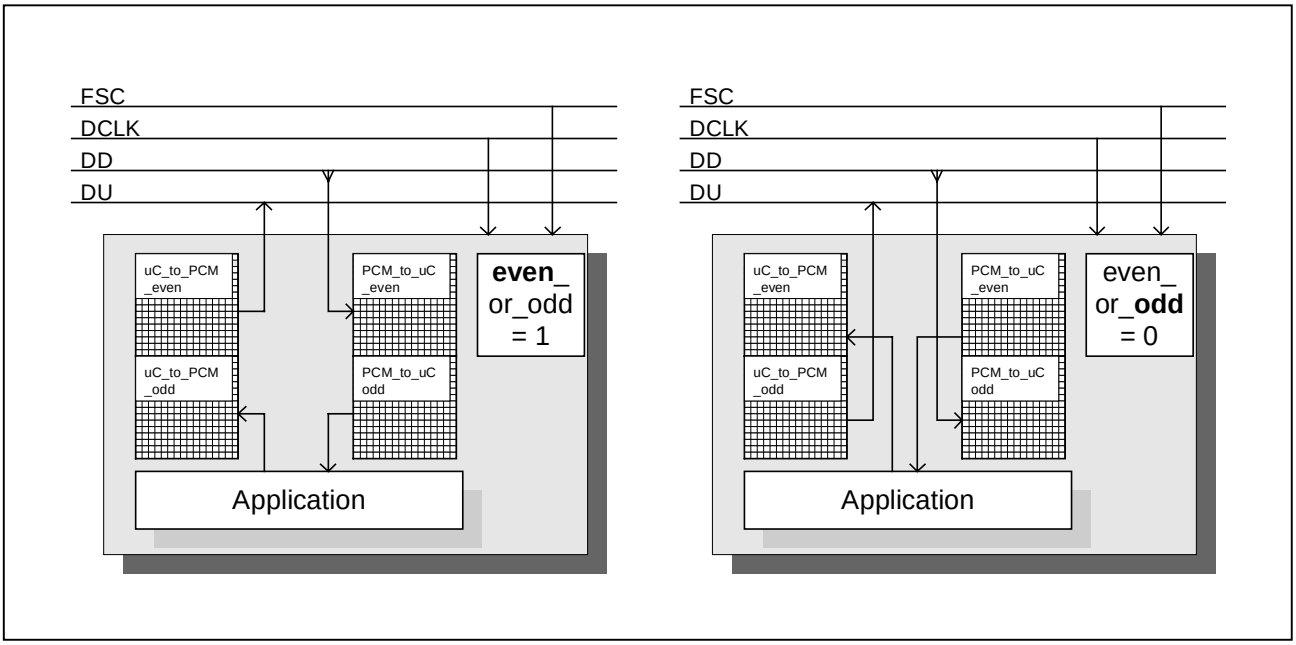

Figure 7 Usage of data buffers

In this application note the buffers are organised 16-bit wide because the SSC allows this number of bits to be transmitted automatically. For the SSC it makes no difference if it sends 32 times 8 bits or only 16 times 16 bits (SSC´s flexibility allows 2 up to 16 bits to be handled). After 256 bits sent / received this solution will stop transmission and set the pointers to the new buffer. With organising the SSC 16-bit wide a minimum of CPU load will be the result.

Estimation of CPU load

As long as the SSC deals with its buffer (by means of the PEC transfer) there is exactly one PEC transfer per word necessary. This results in 16 PEC transfer if one buffer consists of 16 words (assumed, that SSC receives/transmits 16 bits at one time = 2 time slots). So, the transmission/reception of one buffer requires 2 x 16 x 125 ns (one PEC transfer at 16 MHz CPU clock rate needs 125 ns). Factor 2 is necessary, because both has to be done: reception and transmission.

When the SSC has completely filled one buffer (by means of the PEC transfer), the pointers have to be reset to the correct new buffer. This costs additional CPU time. For a very rough calculation it could be assumed that resetting the receive and the transmit buffer takes twice the time a PCM slot needs to be transmitted.

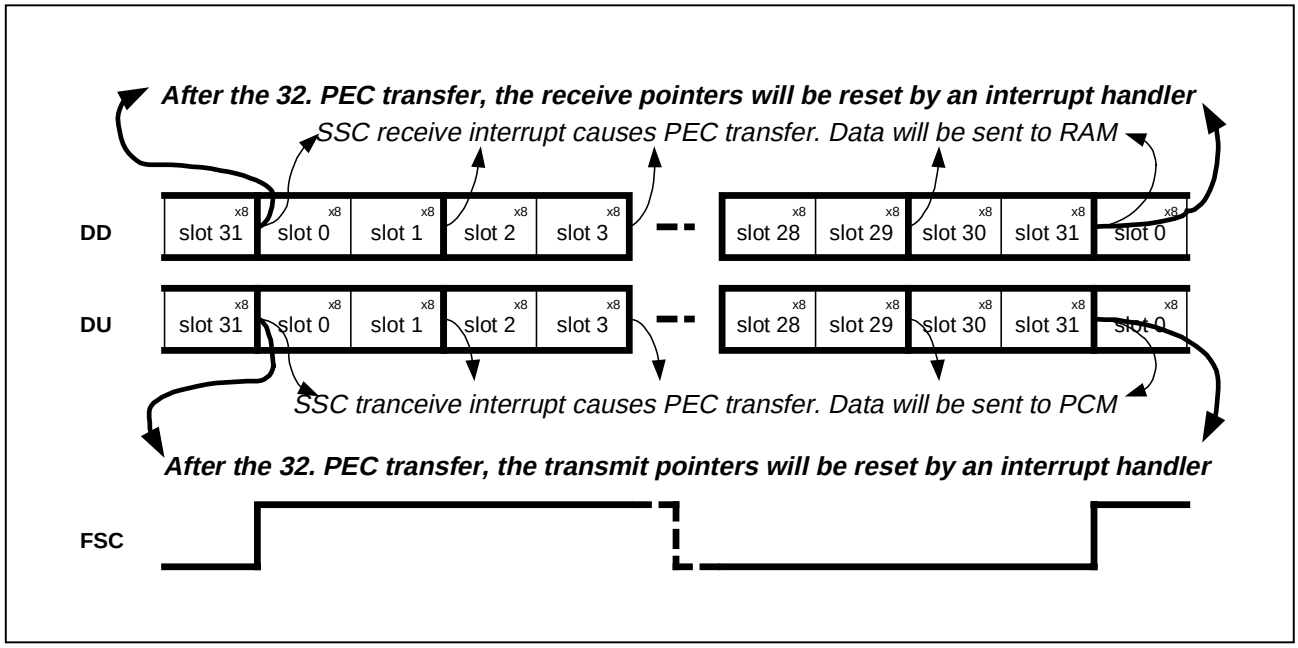

Figure 8 Estimation of CPU load

So, resetting takes 2 x 8 bits x 2.048 Mbit/s = 7.8 us. Adding the CPU time for transmission/reception of 16 words in the buffers (2 x 16 words x 125ns/PEC transfer = 4 us) results in a CPU time of 11.8 us.

Taking into account that one PCM frame takes 125 us (= 32 slots x 8 bits/slot x 2.048 Mbit/s) a CPU load of approx. 10 % can be estimated. Those 10% are necessary for the complete handling of the PCM system. Please refer to chapter 4 for a detailed evaluation.

4 CPU performance and optimisation

Discussing the performance of the different solutions, you should be aware that this discussion is very dependant on the access to the code and data memory. In addition the CPU clock is a second very important factor which influences the performance dramatically.

Code memory

Code access is most efficient if on-chip memory (ROM, Flash, OTP) is used. As the low cost 16-bit microcontrollers from the C16x family usually have no on-chip ROM (IRAM should be reserved for fast data access) the code access is handled by the external bus interface, either by 8-bit or 16-bit code access. In addition this 8 or 16-bit access could be done multiplexed or demultiplexed bus mode. In the summary below numbers are given for a 16-bit demultiplexed, a 16-bit multiplexed and a 8-bit demultiplexed access.

On some C16x devices several Kbytes of XRAM are available on-chip. If time critical code is executed out of that memory area the access is as efficient as zero waitstate code access from a 16 bit demultiplexed external bus.

Data memory

As the access to the data from/to PCM is very time critical, data should be located in the on-chip IRAM. To have an acceptable trade off between “waste” of IRAM and efficient performance 64 or 128 bytes should be allocated to IRAM.

Project	code memory access	CPU clockrate	reset pointers after one PCM frame		PEC transfers for one PCM frame		/us	time for one PCM frame /us		CPU load
PCMuC	16 bit dem.	16 MHz	3.3 + 2.8 us	+	4 us	=	10.1	/125	=	8 %
PCMuC	16 bit dem.	12 MHz	4.5 + 3.9 us	+	5.3 us	=	13.7	/125	=	11 %
PCMuC	16 bit mux.	16 MHz	4.4 + 3.7 us	+	4 us	=	12.1	/125	=	10 %
PCMuCo	8 bit dem.	16 MHz	5.7 + 4.9 us	+	4 us	=	14.6	/125	=	12 %
PCMuCl	16 bit dem.	16 MHz	2.9 + 2.7 us	+	4 us	=	9.6	/125	=	8 %
PCMuCl	16 bit dem.	12 MHz	4.1 + 3.5 us	+	5.3 us	=	12.9	/125	=	10 %
PCMuCl	16 bit mux.	16 MHz	3.8 + 3.4 us	+	4 us	=	11.2	/125	=	9 %
PCMuCl	8 bit dem.	16 MHz	5.1 + 4.3 us	+	4 us	=	13.4	/125	=	11 %

Conditions: code off chip, data buffers on chip (IRAM)

Result

The most efficient way to access the PCM system is by a 16-bit wide demultiplexed bus. In the summary above the solution without feedback (PCMuCl, see chapter 8, 9) and the solution with feedback (PCMuC, PCMuCo, see chapter 8, 9) have been evaluated. In addition different clock speeds and bus modes have been taken into account. It can be said that the PCM highway can be interfaced with a CPU load of approximately 10%. This is a quite acceptable performance.

5 Development Tools

The development of the software modules for PCMuC/PCMuCo and PCMuCl is done by Keil uVision2 V2.00. This version includes the following tools:

- C compiler V4.00a
- Assembler V4.0
- Linker/Locator V4.00
- Librian V4.10
- Hex Converter V4.00

The project files are called PCMuC.uv2, PCMuCo.uv2, PCMuCl.uv2.

For the development the ICE from Hitex DPROBE was used. The DPROBE was directly connected to the signals from / to PCM highway. These signals were generated by the SMART 2000 mainboard / SMART 2100 linecard adapter board including WinEASY control software (http://www.infineon.com/products/ics/33/33.htm -> Tool Support -> SMART 2000, 2100).

Finally SMART 2100 supplies the signals FSC and DCLK for a single clocked PCM signal at 2.048 Mbit/s.

For the test without an emulator the kitCON161 starter kit was used. This board was equipped with C161RO at 5V power supply.

6 Support of kitCON161 (platform for demonstration hardware)

General

For evaluation of the code a very price sensitive approach was chosen by using the starter kits (kitCON) of C16x microcontroller family (http://www.infineon.com/products/ics/34/index2.htm -> Starter Kits -> SK 166 Starter Kit). Here, the starter kit for C161V/K/O is used. This board is equipped with a 16 MHz crystal (= 16 MHz CPU clock rate at direct drive) and it is possible to select either 16-bit demultiplexed code access or 16-bit multiplexed code access. Of course, the 16 MHz crystal may be replaced by any different clock source.

Together with a source sending the 2.048 Mbit/s DCLK clock signal and the start of the frame signal FSC a nice test environment can be set up. Hint: the source sending DCLK and FSC is not part of the starter kit. (For our evaluations we used the SMART 2000 mainboard / SMART 2100 linecard adapter board together with WinEASY control software (http://www.infineon.com/products/ics/33/33.htm -> Tool Support -> SMART 2000, 2100). Finally SMART 2100 supplies the signals FSC and DCLK for a single clocked PCM signal at 2.048 Mbit/s.)

To test the quality of the prepared application note a short user specific test routine could be added to the delivered code. Finally, this complete test sequence can be downloaded to the on-board flash memory of the starter kit. After finishing the process of downloading the complete application could be checked by a tool which scans the PCM bus continuously.

In the evaluation software of this application note C16x sends a certain pattern to one PCM slot continuously. By linking data upstream (DU) and data downstream (DD) together this pattern will be received by C16x and only one device in the PCM system is necessary.

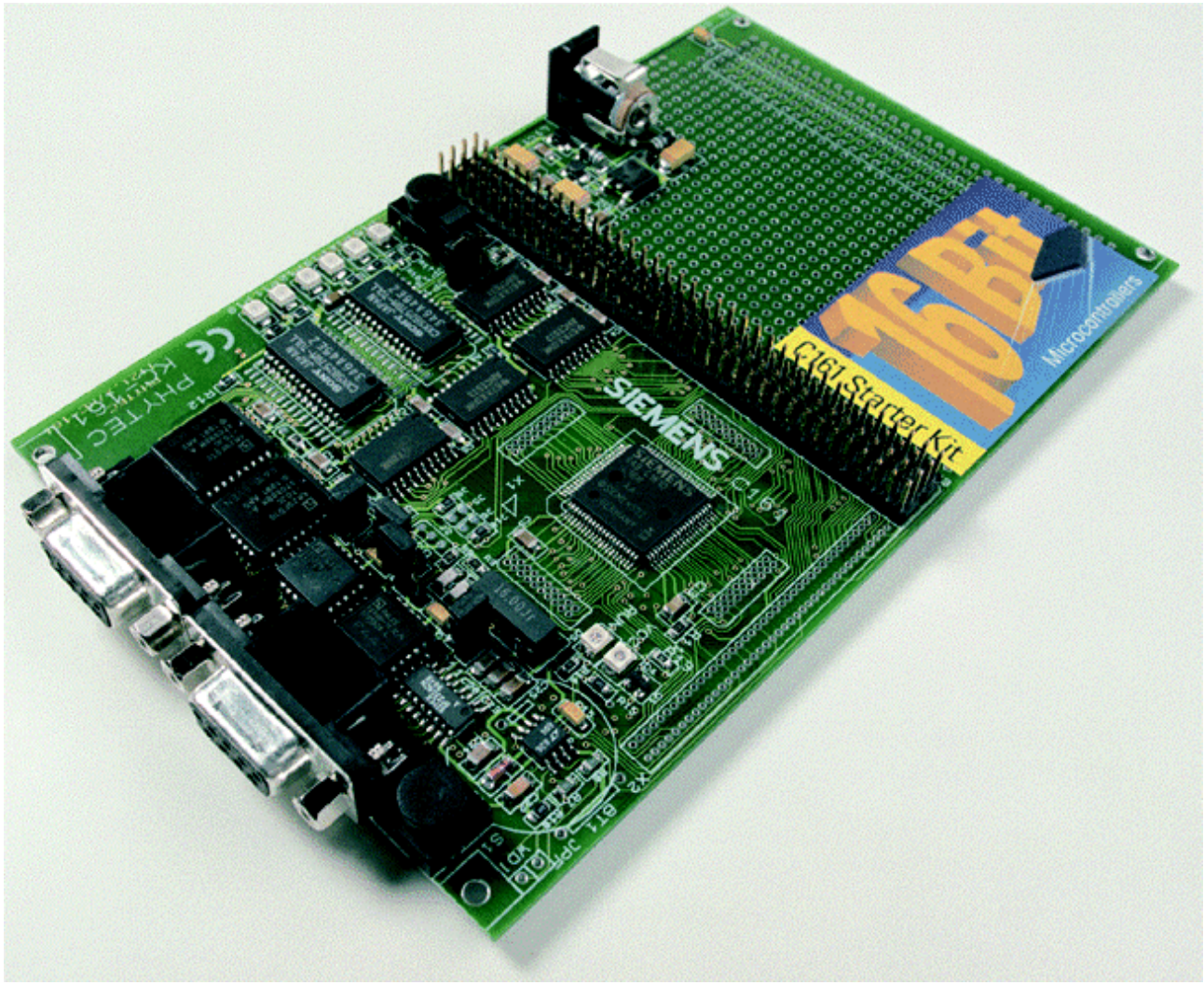

Figure 9 C161 kitCON starter kit

Details

The following default settings should be used for the kitCON161:

JP1	2+3	selects 16-bit demultiplexed code access (1+2: multiplexed)
JP2	1+2	no on-chip code available
JP3	2+3	RAM memory size
JP4	open	Memory mode 1 (Flash at Adr. 0)
JP9	2+3	normal program execution
J10	closed	direct drive (open: clock/2)

To download code to the on board flash the following jumpers have to be set:

JP9	1+2	start external bootstrap loader (not C161 on-chip loader!)

Using the Phytec Flash Tools the application may now be downloaded to the on board flash. Those tools may be found on the CD which is delivered with the kitCON package. (\CDROM\STARTKIT\C161\FLASH\flasht). For details, please check example 2 at “Getting started”.

To download your code the RS232 cable has to be connected to that DSUB connector at kitCON which is close to the printed “PHYTEC”-logo on board. Before you start flash tools do not forget to reset your board.

After successfully downloading the code the jumper setting has to be changed again:

JP9	2+3	normal program execution

Now, with pressing the reset button, your application should be executed out of the on board flash memory. But without having applied the application specific signals, nothing will happen. So, to use the code attached to this application note the following signals have to be connected:

Connections at kitCON connector:

Pin Nr. kitCON connector	94	109	113	115	95
Function at C161O	EX2IN	MRST	MTSR	SCLK	Output
Port at C161O	P2.10	P3.8	P3.9	P3.13	P2.12
connect externally to:	FSC	DU, pull up	DD, pull up	DCLK	optional test pin
Oscilloscope for evaluation	Ch1	Ch3 (trigger)	Ch3 (trigger)	Ch2	Ch4

DU and DD should be connected together to use the transmitted data stream for reception.

The signals DCLK and FSC should be applied correctly from external. With pressing reset a second time C161O should synchronise to the PCM highway and send the right pattern in the right slot. This may be checked by an oscilloscope.

7 Details of PEC response times

At the implementation of the PCM time-division-multiplex interface the PEC response time is a very critical point. This PEC response time influences the delay (slippage) between the occurrence of FSC rising edge, detecting that edge at the fast external interrupt pin EX2IN, switching on the SSC by a PEC transfer and finally transmitting / receiving the first bit by the PCM time-division-multiplex interface.

The following section gives an idea about the number of CPU states which are necessary. The detection of FSC rising edge at pin EX2IN takes some states. This fast external interrupt request triggers a PEC transfer. In this PCM application this injected PEC transfer follows an instruction which is executed out of the jump-cache of the pipeline (see endless loop in main.c). Alternatively one may use the IDLE state of the CPU, which is followed by the PEC transfer. Attention: the source pointer of the EX2IN-PEC should be in IRAM. Finally, the decoding and execution of the PEC transfer (switches SSC on) requires some additional states. These delays result in a total of 8..10 states if the endless loop is programmed and 9..10 states if the IDLE state is programmed. In the software attached to this application note only the endless loop is implemented.

The following overview discusses the situation if the CPU is driven with different clock rates. At 20 MHz and 16 MHz, the slippage is exactly one bit. At 12 MHz the slippage will be 1 or 2 bits if an endless loop is used and exactly 2 bits if IDLE state is used. With a CPU running at 10 MHz, the slippage will be two bits.

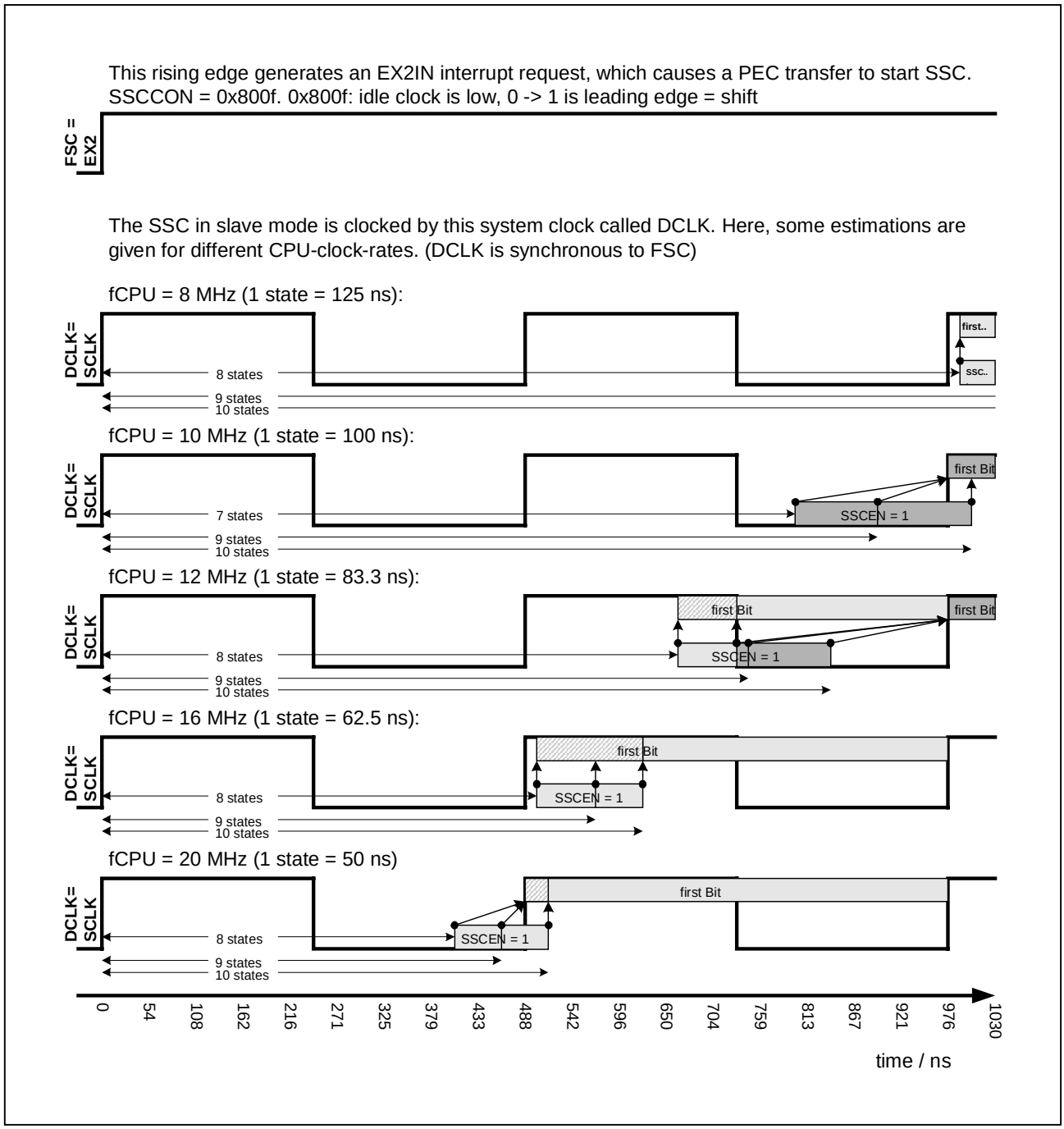

Figure 10 Slippage at different CPU speeds

In the SSC it is important if the SSCEN bit is set during the high phase of SCLK or during the low phase of SCLK. If it is set during the low phase (here: the idle clock is low), then the SSC starts transmission with the next rising edge of SCLK. If it is set during the high phase, then the SSC transmits immediately the first bit and starts at the next rising edge of SCLK with the second bit of the serial bit stream.

8 Details of implementation

8.1 System without feedback from an external partner. C16x verifies the slippage by itself (PCMuCo is an optimised version of PCMuC)

General

There are two versions implemented for systems which give no feed back. On those systems C16x has to check by itself if it is synchronised properly. The two software versions PCMuC and PCMuCo differ only in the level of code optimization for execution time.

Both versions start the synchronisation by waiting for a rising edge at FSC which triggers the start of the C16x SSC in slave mode (PEC transfer). Because of a certain delay in the C16x the reception of the incoming bitstream will start with a certain slippage. This slippage will be obvious when the next FSC occurs. By linking the FSC event to a start condition of a timer and the reception of one slot to a stop condition the slippage may be calculated. This is done by the C16x on-chip timer T5.

After this process of verification the C16x needs some FSCs to restart the transmission or reception based upon the calculated slippage. The C16x has now to take care about correctly meeting the right slot of the PCM system.

With the code of PCMuC / PCMuCo the pattern 0x08 is transmitted in PCM timeslot 6 and timeslot 7 (start counting timeslots with 0!) with LSB first. (The transmission/reception will occur with a constant slippage). So the data will not be found exactly at that slot but one bit shifted. The data to be sent via PCM may be changed in the module main.c. In the code of PCMuC/PCMuCo data from PCM at slot 6 and slot 7 will be stored in the receive buffer at the 7^{th} and 8^{th} position (start with 1 to count). As the buffer is organised 16-bit wide, the result can be found at the 4^{th} position, talking in the language of a C-array it will be nr[3]. Data which have to be sent to PCM at slot 6 and 7 have to be stored at the 5^{th} and 6^{th} position (start counting with 1!). In C-language the definition would be byte nr[2] (16-bit wide organised array).

In the application software of PCMuC/PCMuCo only one of the two transmit buffers sends the pattern 0x08 in slot 6 and 7. So, watching the bitstream with the oscilloscope will show the pattern 0x08 only at every second PCM frame.

Optimised version

The two projects PCMuC and PCMuCo differ in that way that PCMuC compiles the file INT.c first to an assembler file called INT.src. Remember: In the file INT.c / INT.src / INTo.src the time critical routines can be found. There the interrupt handler for the fast external interrupt EX2IN is very important. In addition the interrupt handlers for SSC transmit and receive interrupt (SSC.c) are very critical.

After that process of compiling the assembler is used to generate object code out of this INT.src file. The version PCMuCo does not use the C compiler. There a file called INTo.src

which is used directly as source to be assembled. The idea behind this is to have a standard INT.src file from PCMuC which can be optimised regarding the assembler code. After optimisation it has to be renamed to INTo.src and has to be linked to the new project PCMuCo. So, it is quite convenient to handle the different versions of a standard solution (PCMuC) together with an optimised solution (PCMuCo).

In our case pushing of a register (R4) was deleted. There are additional features which could be implemented to have a more code or speed optimised solution:

In both solutions there is no saving of the data page pointers (DPP). This is a result of the C-compiler flag *NODPPSAVE* . There are applications where this flag must not be set.

A working register like R4 or R6 can be dropped as well, if variables which are transported to the core are linked directly with registers which are not used by the application.

Finally the buffer which stores the slots to be transmitted / received may be larger or smaller to get both items right balanced: performance and waste of IRAM memory area.

8.2 System with feedback from an external partner (PCMuCl)

The version PCMuCl is very similar to PCMuC/PCMuCo. In those applications the C16x does not verify the slippage by itself. In those applications the verification may be done if the contents of one PCM slot is well defined. Then the C16x may compare the expected result and the actual received byte.

With the code of PCMuCl the pattern 0x08 is transmitted in PCM timeslot 4 and timeslot 5 (start counting timeslots with 0!) with LSB first. This may be changed in the module main.c.

With the code of PCMuCl the pattern 0x08 is transmitted in PCM timeslot 8 and timeslot 9 (start counting timeslots with 0!) with LSB first. Have in mind, that the transmission/reception will occur with a constant slippage. So the data will not be found exactly at that slot but one bit shifted. The data to be sent via PCM may be changed in the module main.c. At the code of PCMuCl data from PCM at slot 8 and slot 9 will be stored in the receive buffer at the 9^{th} and 10^{th} position (start with 1 to count). As the buffer is organised 16-bit wide, the result can be found at the 5^{th} position, talking in the language of a C-array it will be nr[4]. Data which have to be sent to PCM at slot 8 and 9 have to be stored at the 5^{th} and 6^{th} position (start counting with 1!). T In C-language the definition would be byte nr[2] (16-bit wide organised array).

In the application software of PCMuCl only one of the two transmit buffers sends the pattern 0x08 in slot 8 and 9. So, watching the bitstream with the oscilloscope will show the pattern 0x08 only at every second frame.

9 Timing diagrams

PCMuC / PCMuCo

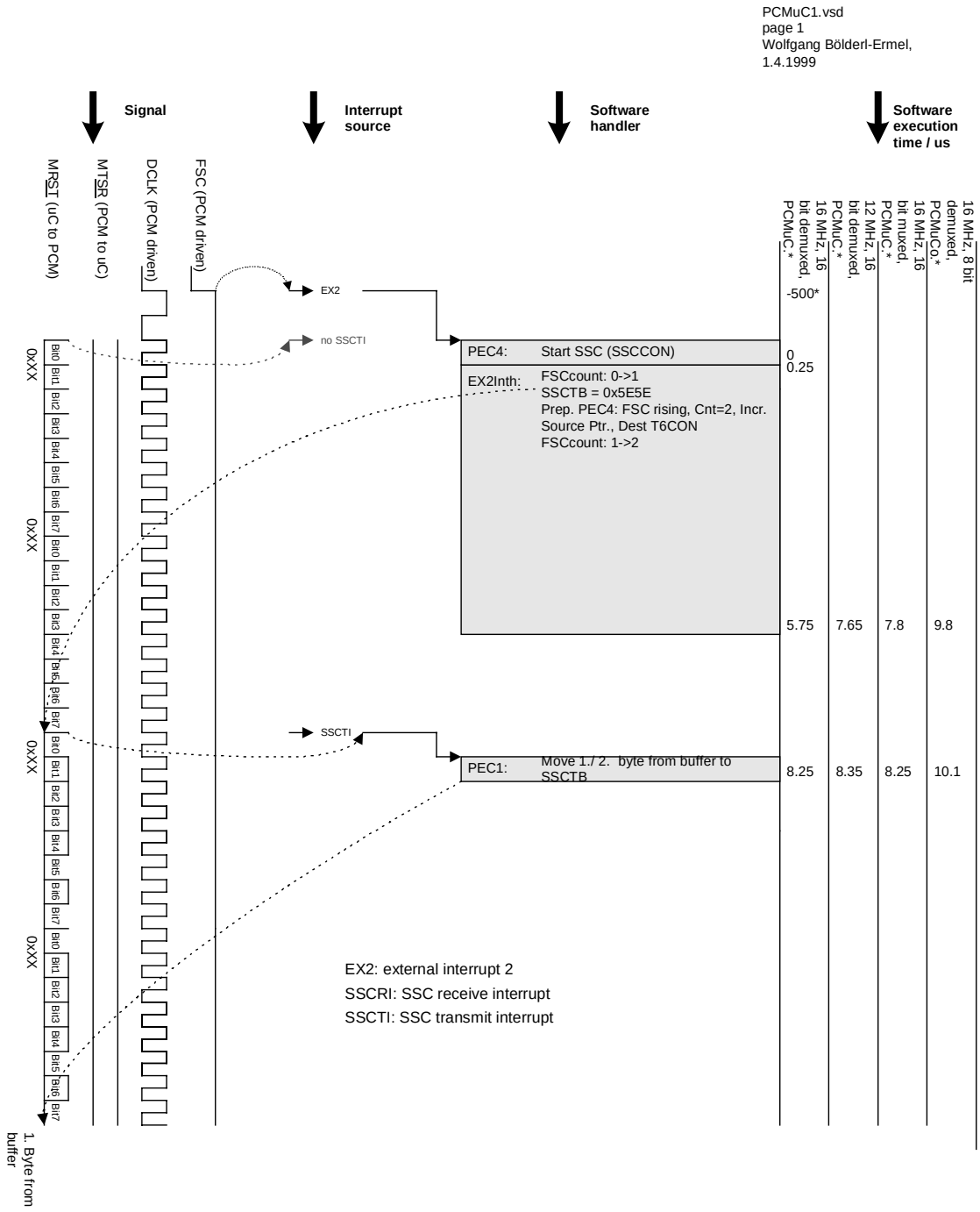

Figure 11 Timing (1) PCMuC / PCMuCo

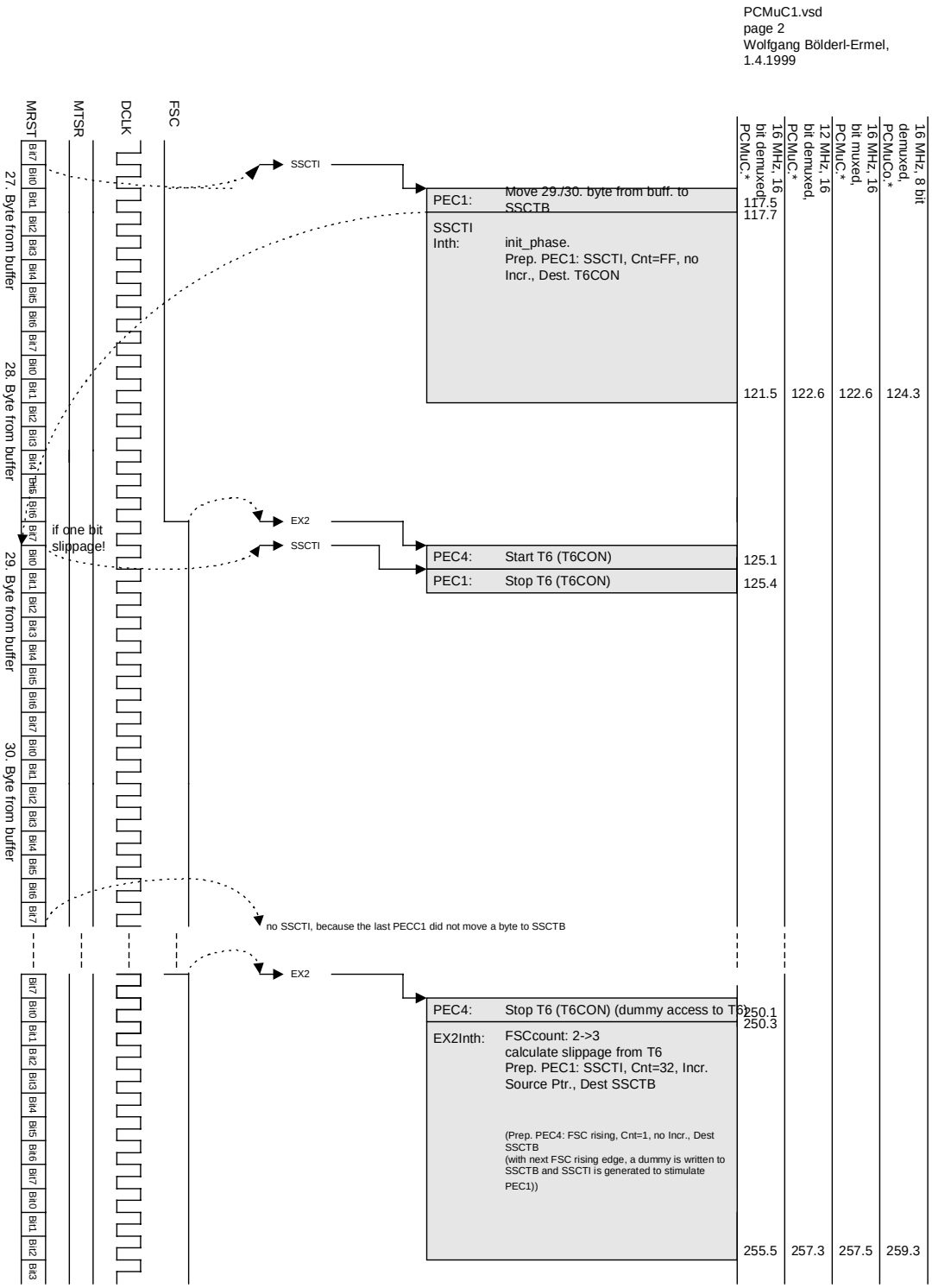

Figure 12 Timing (2) PCMuC / PCMuCo

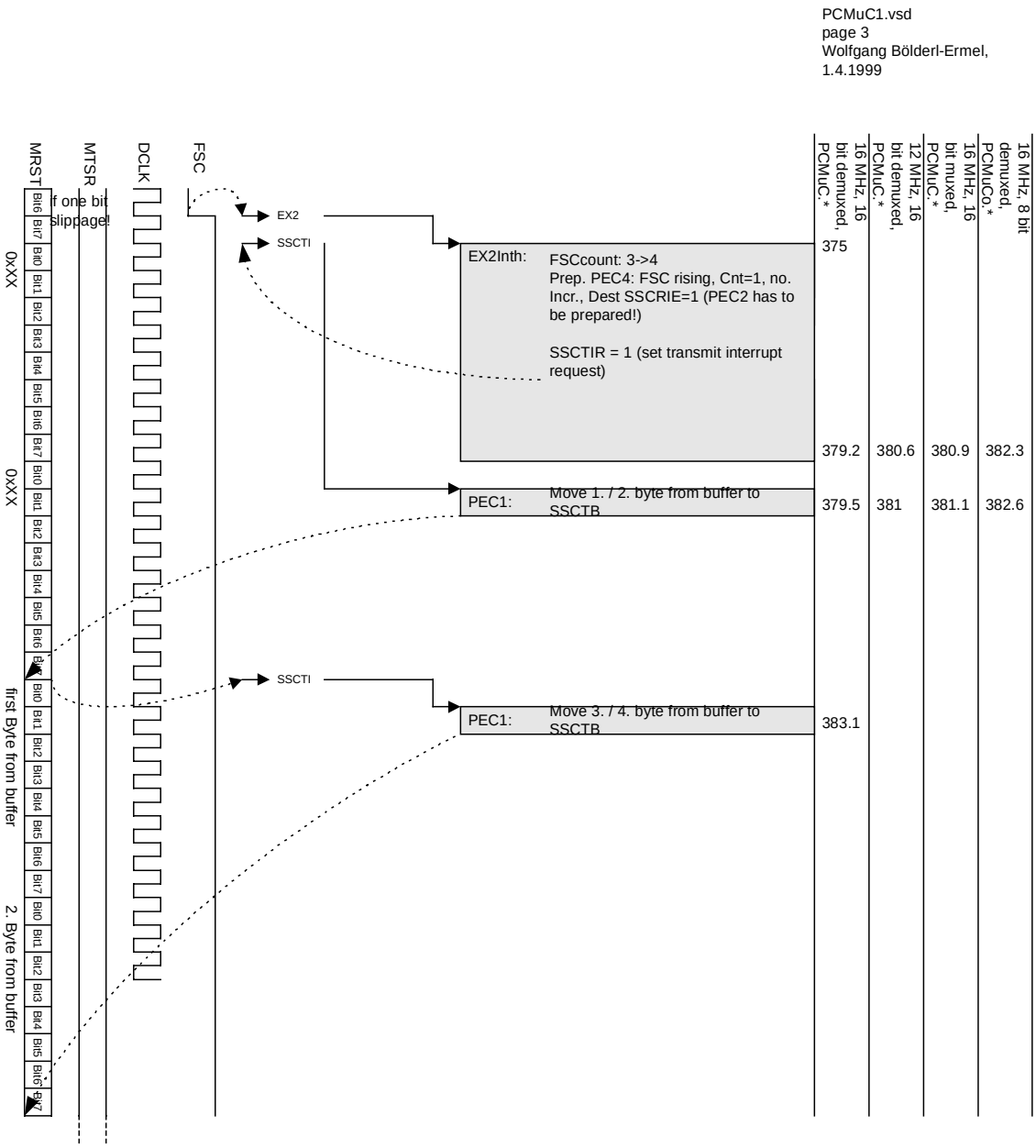

Figure 13 Timing (3) PCMuC / PCMuCo

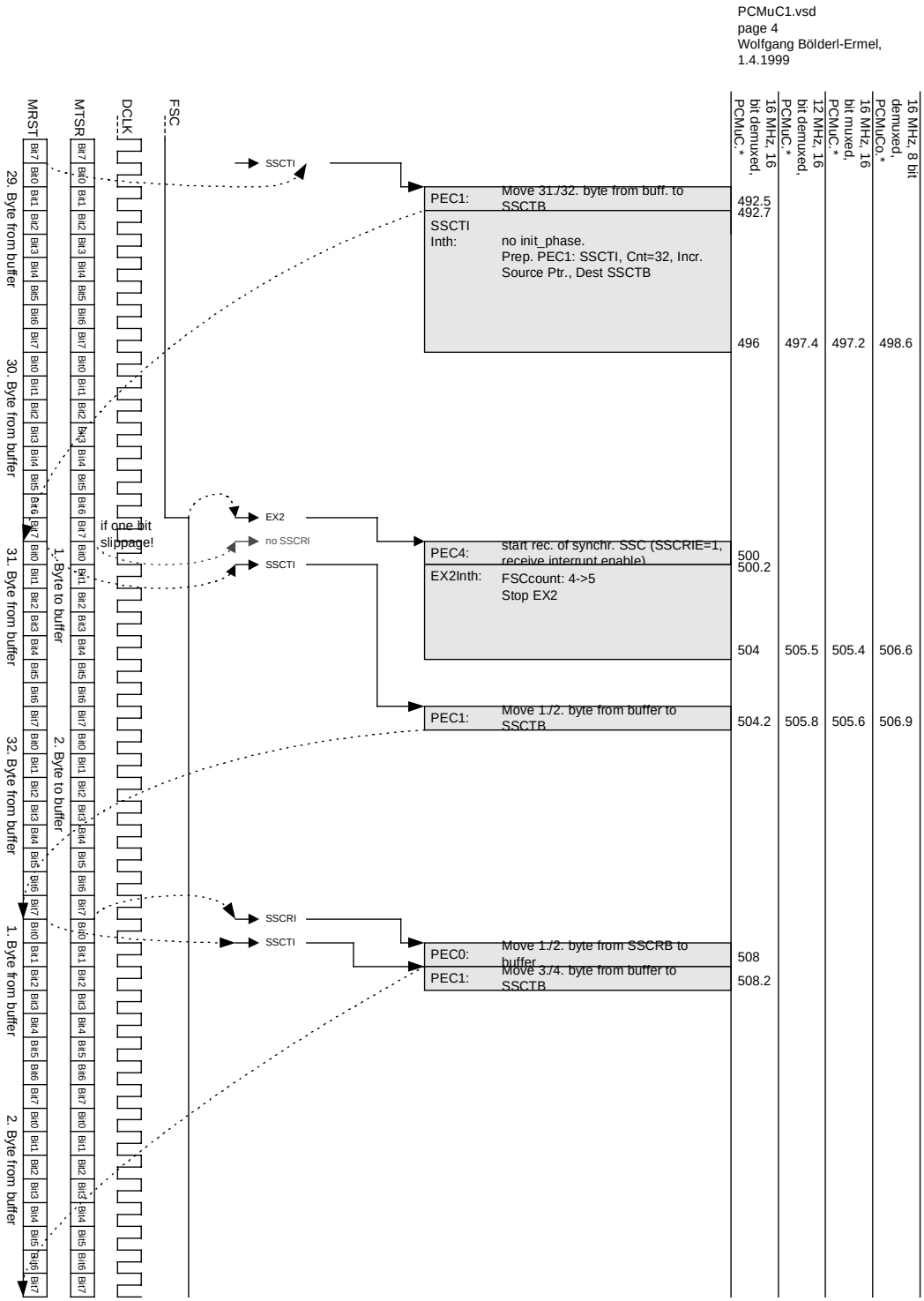

Figure 14 Timing (4) PCMuC / PCMuCo

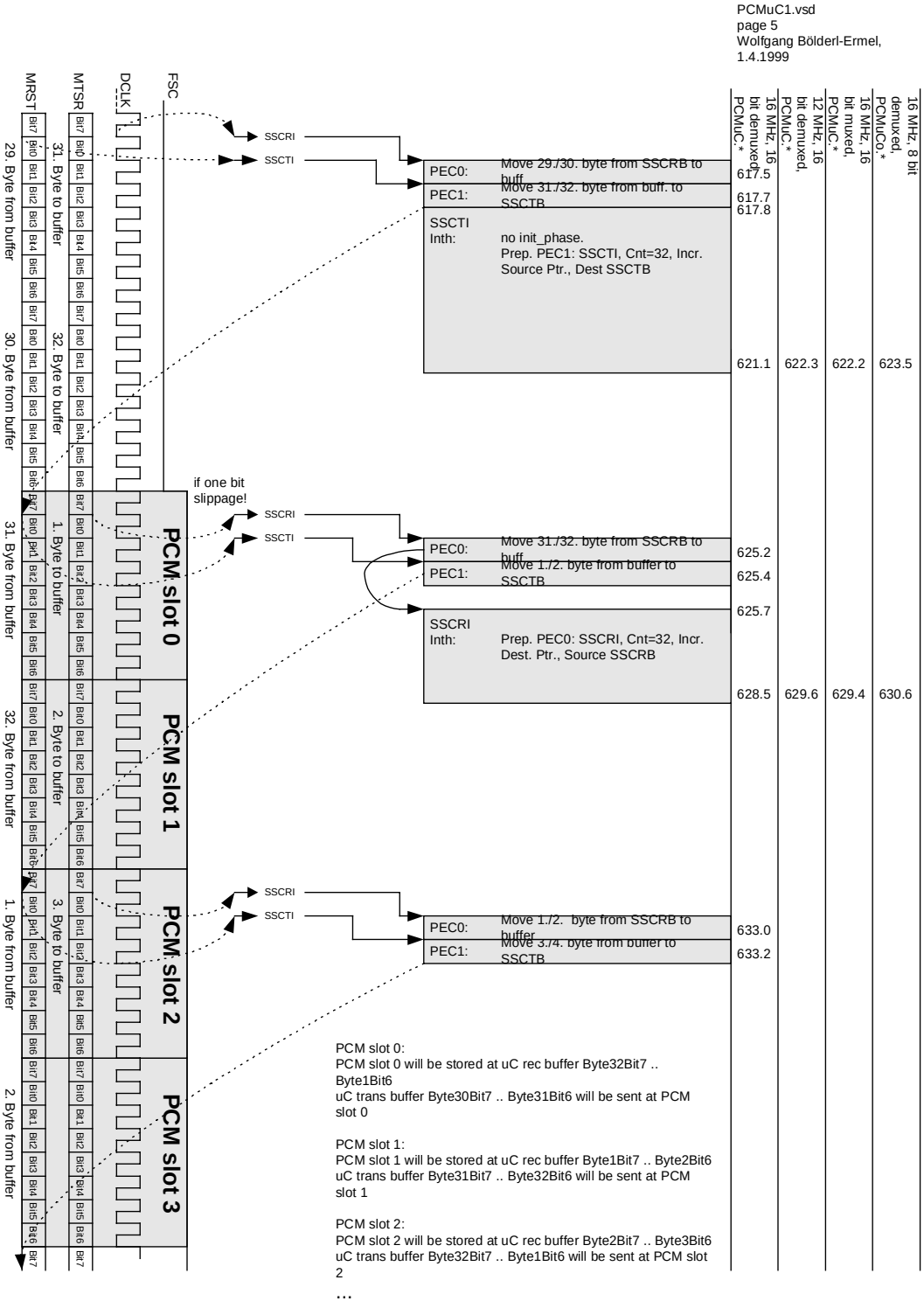

Figure 15 Timing (5) PCMuC / PCMuCo

PCMuCl:

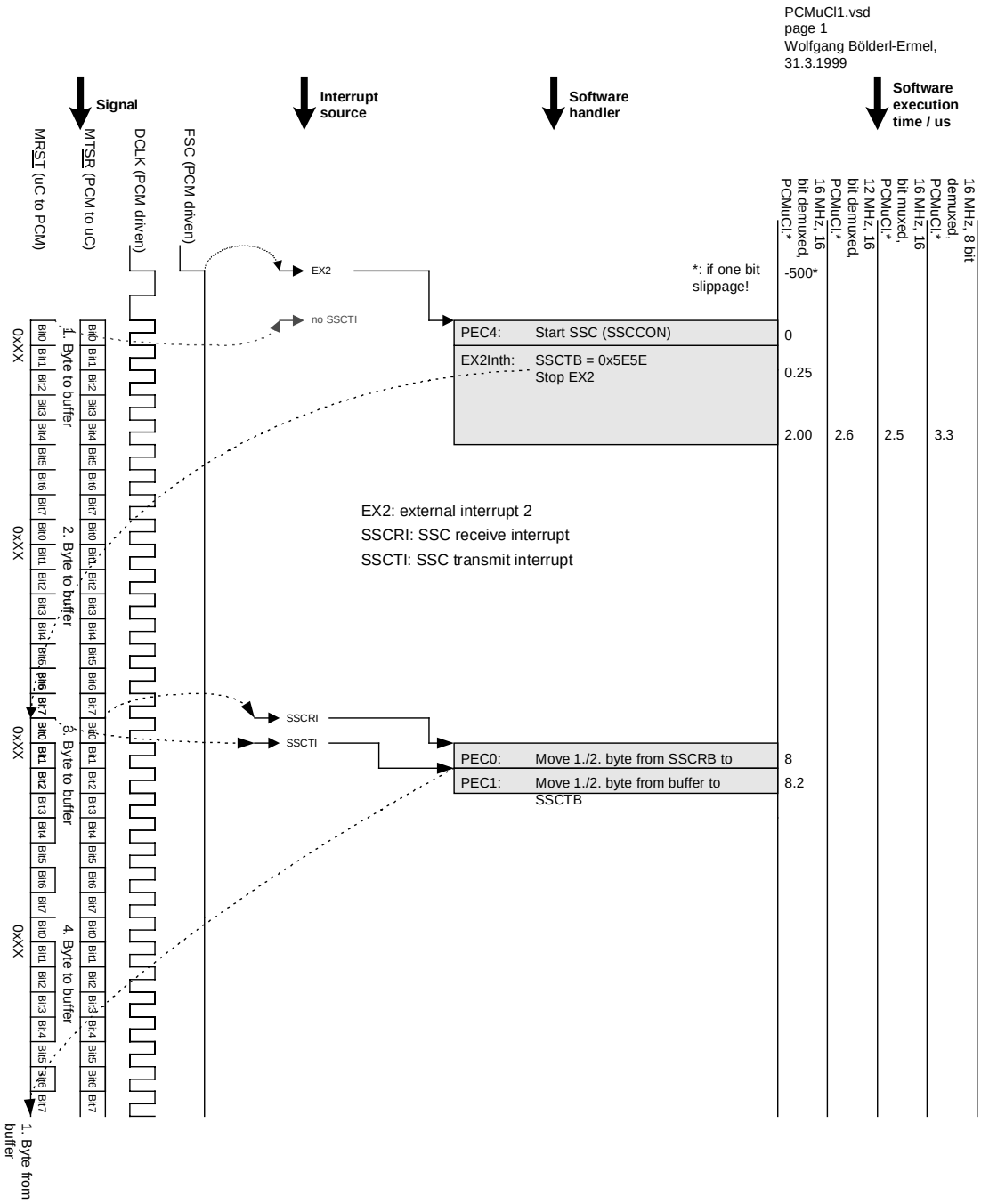

Figure 16 Timing (1) PCMuCl

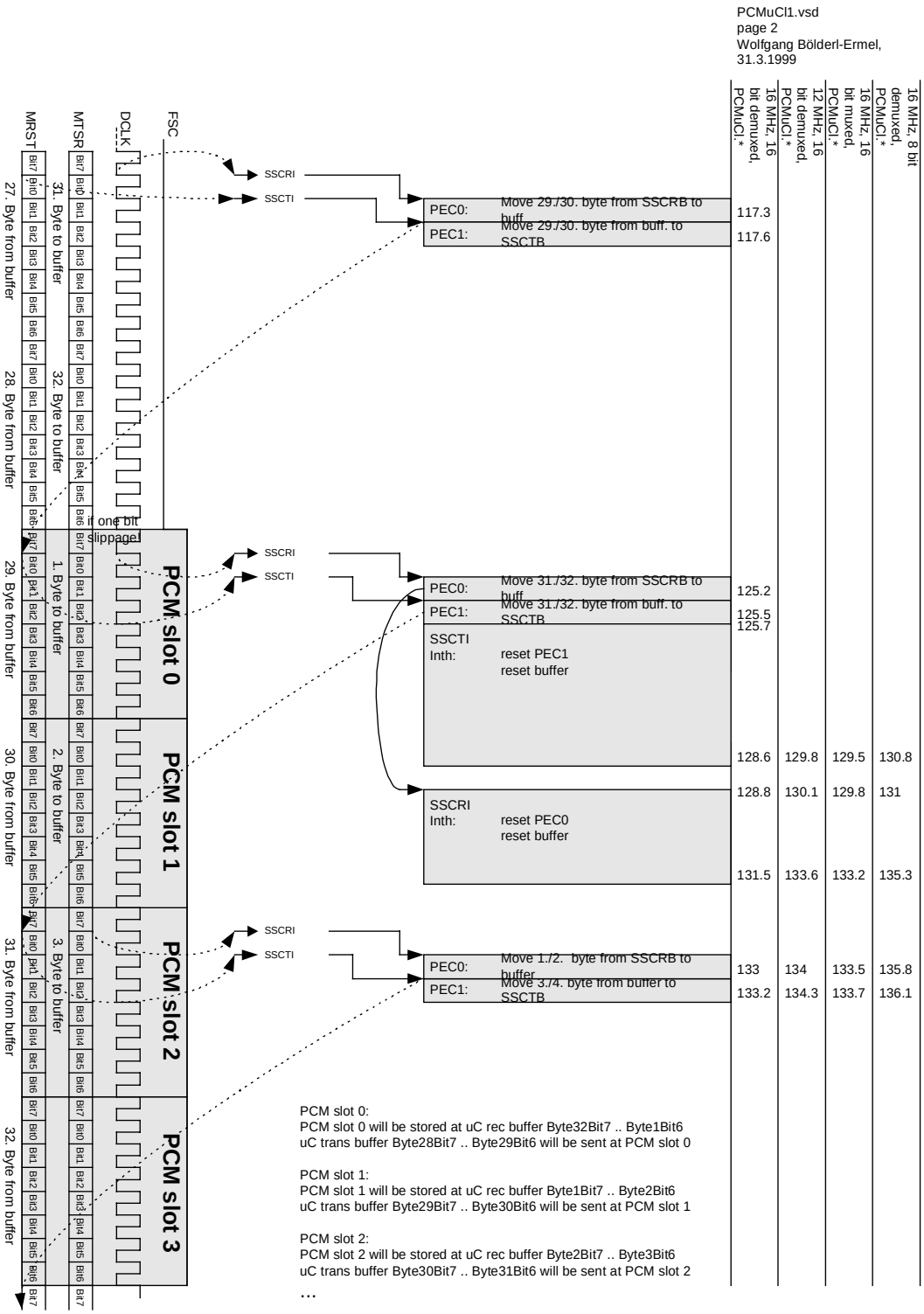

Figure 17 Timing (2) PCMuCl