Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

User Manual, V 1.7, January 2001

C166S V2

16-Bit Microcontroller

Microcontrollers

Infineon

Never stop thinking.

Edition 2001-01

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
D-81541 München, Germany

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

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.

C166S V2

16-Bit Microcontroller

Microcontrollers

Infineon

Never stop thinking.

C166S V2

Revision History:	**2001-01**	V 1.7
Previous Version:	-	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

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

ce.cmd@infineon.com

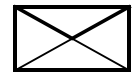

Table of Contents **Page**

Table of Contents **Page**

1 Introduction

C166S V2 is a member of the most recent generation of the popular C166 microcontroller cores. C166S V2 combines high performance with enhanced modular architecture. It was developed to provide easy migration from standard existing C16x to the new C166S V2 core with its impressive DSP performance and advanced interrupt handling. The system architecture inherits successful hardware and software concepts that have been established in the C16x 16-bit microcontroller families. C166 code compatibility enable re-use of existing code. This dramatically reduces the time-to-market for new product development.

The following features position C166S V2 strategically for contemporary and emerging markets for performance-hungry real-time applications:

– High CPU performance. Single clock cycle execution doubles the performance at the same CPU frequency (relative to the performance of the C166).
– Built-in advanced MAC unit dramatically increases DSP performance.
– High Internal Program Memory bandwidth and the instruction fetch pipeline significantly improve program flow regularity and optimize fetches into the execution pipeline.
– Sophisticated Data Memory structure and multiple high-speed data buses provide transparent data access (0 cycles) and broad bandwidth for efficient DSP processing.
– Advanced exceptions handling block with multi-stage arbitration capability yields stellar interrupt performance with extremely small latency.
– Upgraded Peripheral Event Controller supports efficient and flexible DMA features to support a broad range of fast peripherals.
– Highly modular architecture and flexible bus structure provide effective methods of integrating application-specific peripherals to produce customer-oriented derivatives.

This User's Manual describes the new standard C166S V2 core independently from its use for the dedicated product. Differencies to existing standard products are therefore described in the User's Manual (or Target Specification) of the product.

1.1 Technical Overview

– 5-stage execution pipeline
– 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
– Pipeline with forwarding that controls data dependencies in hardware
– Linear address space for code and data (von Neumann architecture)
– Multiple high bandwidth internal busses for data and instructions
– Enhanced memory map with extended I/O areas
– 16 MBytes total linear address space
– C16x family compatible on-chip special function register area
– Fast multiplication (16-bit x 16-bit) in one CPU clock cycle
– Fast background execution of division (32-bit/16-bit) in 21 CPU clock cycles

– Nearly all instructions executed in one CPU clock cycle
– Enhanced boolean bit manipulation facilities
– Zero cycle jump execution
– Additional instructions to support High Level Language (HLL) and operating systems
– Register-based design with multiple variable register banks
– Two additional fast register banks
– General purpose register architecture
– 16 General-purpose registers (GPRs) for byte operands
– 16 General-purpose registers (GPRs) for integer operands
– Overlapping 8-bit and 16-bit registers
– Opcode fully upward compatible with C166 family
– Variable stack with automatic stack overflow/underflow detection
– High performance branch-, call- and loop processing
– Multiply and accumulate instructions (MAC) executed in one CPU clock cycle
– Extremely short interrupt response time
– "Fast interrupt" and "Fast context switch" features
– Peripheral bus (PDBUS+) with bit protection

1.2 System Description

The basic C166S V2 System consists of the following main units:

- C166S V2 CPU
- On-Chip Data- and Code-Memories
- Data Management Unit (DMU)
- Program Management Unit (PMU)
- Interrupt and Peripheral Event Controller (PEC) Controller
- OCDS and JTAG-Interface
- External Bus Controller (EBC)
- System Control Unit (SCU)
- Clock Generation Unit (CGU)

The powerful C166S V2 core, the peripherals, and the internal memories of the C166S V2 microcontroller are connected to various busses:

- 16-bit high performance system bus
- 16-bit enhanced peripheral bus (PDBUS+)
- 64-bit internal program memory bus
- 16-bit data memory bus

Figure 1-1 shows a typical configuration of a C166S V2-based system.

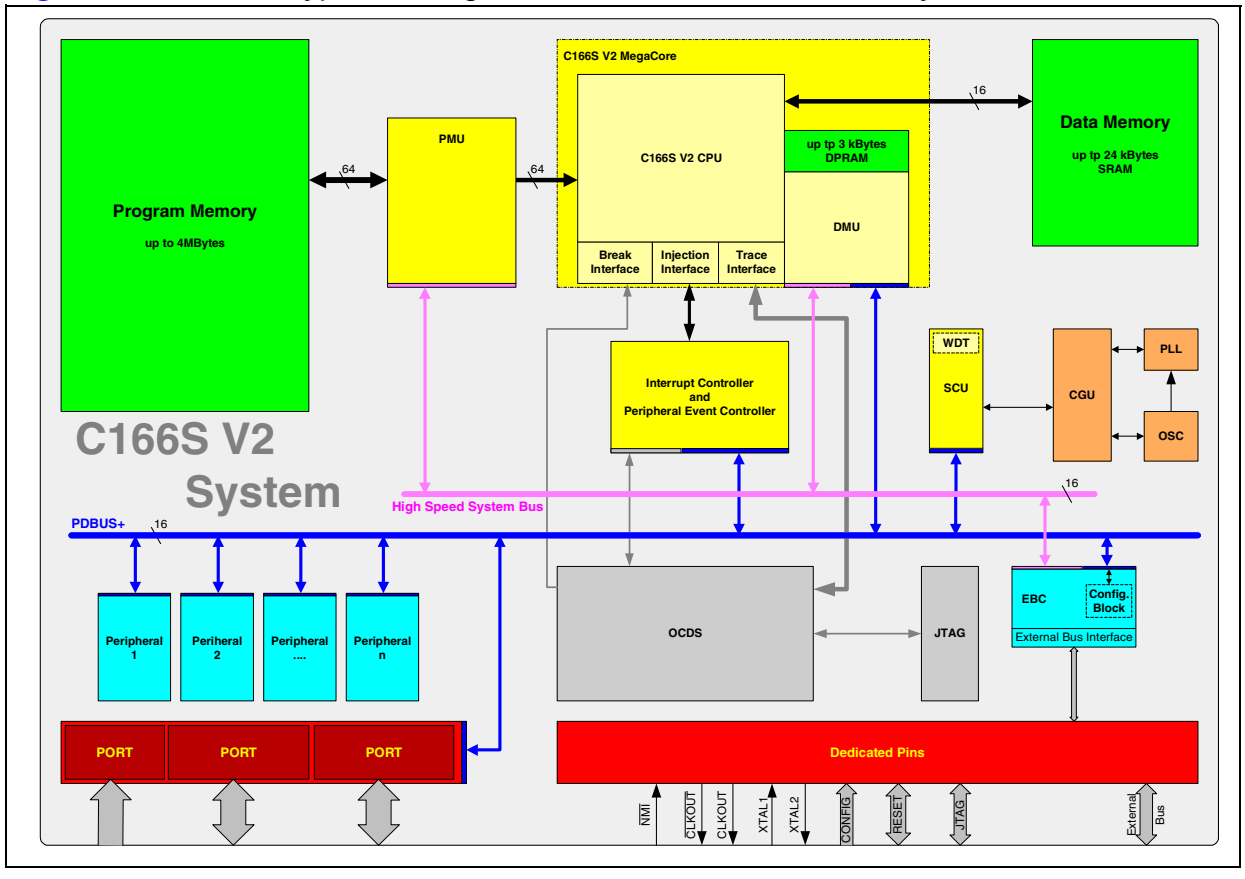

Figure 1-1 C166S V2 System

1.2.1 CPU

- 5-stage execution pipeline
- 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
- Pipeline with forwarding that controls data dependencies in hardware
- Flexible PMU and DMU with cache capabilities
- Linear address space for code and data (von Neumann architecture)
- Multiple high bandwidth internal busses for data and instructions
- 16 MBytes total linear address space
- Nearly all instructions executed in one CPU clock cycle
- Enhanced boolean bit manipulation facilities
- Zero cycle jump execution
- Additional instructions to support HLL and operating systems
- Register-based design with multiple variable register banks
- Two additional fast register banks
- General purpose register architecture
- 16 General-purpose registers (GPRs) for byte operands
- 16 General-purpose registers (GPRs) for integer operands

– Overlapping 8-bit and 16-bit registers

Multiply Accumulate Unit (MAC)

– Single cycle MAC with zero cycle latency including a 16*16 multiplier plus 40-bit barrel shifter; single clock multiplication is ten times faster than C166 at the same CPU clock
– 40-bit accumulator to handle overflows
– Automatic saturation to 32 bit or rounding included with the MAC instruction
– Fractional numbers supported directly
– One Finite Impulse Response Filter (FIR) tap per cycle with no circular buffer management

1.2.2 On-Chip Memory Modules

– Up to 3 KBytes on-chip dual ported SRAM for DSP data and register banks
– Up to 24 KBytes on-chip internal single ported SRAM module for data storage
– Up to 4 MBytes on-chip memory module for program storage

Note: The on-chip memory configuration may differ from product to product. Product specific on-chip memory configurations are defined in the corresponding product specifications.

1.2.3 Data Management Unit (DMU)

The Data Management Unit (DMU) handles all data transfers external to the core (i.e. external memory or on-chip special function registers on the PDBUS+) and instruction fetches in external memory. The DMU acts as a data mover between the various interfaces. By handling all these interfaces, it incorporates the C166S V2 System Bus. An access prioritization between **E**xternal **B**US **C**ontroller (EBC) accesses from the core and **P**rogram **M**emory **U**nit (PMU) is handled by the DMU. This allows an instruction fetch from external memory in parallel with data access that is not on EBC.

1.2.4 Program Memory Unit (PMU)

The PMU has two basic functions: to provide the CPU with instructions and to provide the CPU (through the DMU) with data located in the Internal Program Memory. The Internal Program Memory is implemented within the PMU.

The instructions requested by the CPU can be located in the Internal Program Memory; in which case, the instructions are requested to the internal memory. Alternatively, they can be located in external memory; in which case, the PMU re-sends this request to the EBC through the DMU, receives the data from the external memory, through the EBC/DMU, and delivers it as the requested instruction to the CPU.

1.2.5 Interrupt and PEC Controller

– 16-Priority-level interrupt system with up to 128 sources on four group levels
– Eight PEC channels with 24-bit source and destination pointers with segment pointer registers
– Enhanced PEC pointers. PEC source pointers and PEC destination pointers can be simultaneously modified
– Independent programmable PEC level and "End of PEC" interrupt

1.2.6 OCDS and JTAG

The OCDS (level 1) provides facilities to the debugger to emulate resources and assist in application program debug. The main features are:

– Real time emulation
– Extended trigger capability including: instruction pointer events, data events on address and/or value, external inputs, counters, chaining of events, timers, etc.
– Software break support
– Break and “break before make” (on IP events only)
– Interrupt servicing during break or monitor mode
– Simple monitor mode or JTAG based debugging through instruction injection

The C166S V2 OCDS is controlled by the debugger[1)] through a set of registers accessible from the JTAG interface. The OCDS also receives informations (such as IP, data, status) from the core for monitoring the activity and generating triggers. Finally, the OCDS interacts with the core through a break interface to suspend program execution, and through an injection interface to allow execution of OCDS generated instructions.

1.2.7 External Bus Controller (EBC)

All external memory accesses are performed by a particular on-chip External Bus Controller (EBC).

1.2.8 System Control Unit (SCU)

The System Control Unit supports all central control tasks and all product specific features. The following typical sub-modules are implemented in this unit:

Reset Control

The reset function is controlled by the reset control unit.

1) Debugger refers to the tool connected to the emulator, and more specifically to the OCDS via the JTAG and which manages the emulation/debugging task.

Power Saving Control

The Power Saving Control block, known from the power management of the C166 derivatives, manages idle mode, power down mode, and sleep mode of the C166S V2.

ID Control

A set of six identification registers is defined for the most important silicon parameters, including the chip manufacturer, the chip type and its properties. These ID registers can be used for automatic test selection.

External Interrupt Control

The C166S V2 System provides asynchronous fast external interrupt inputs.

Central System Control

The central system behavior of the C166S V2 is controlled by this block. The frequency of the PDBUS+ (bus clock) and of all peripherals connected to this bus is programmable according to the maximum physical bus speed and the application requirements. Furthermore, the clock generation status is indicated. Depending on the application state, various security levels (such as protected and unprotected mode) are supported by the security level control state machine.

Watchdog Timer (WDT)

The Watchdog Timer is one of the fail-safe mechanisms that have been implemented to prevent the controller from malfunctioning. However, the Watchdog Timer can detect only long term malfunctions.

1.2.9 Clock Generation Unit (CGU)

The C166S V2 Clock Generation Unit uses either an oscillator or crystal to generate the system clock. A programmable on-chip PLL adds high flexibility to clock generation for the C166S V2.

1.2.10 On-Chip Bootstrap Loader

As in the C166, the on-chip bootstrap loader allows the start code to be moved into internal RAM via the serial interface.

2 Central Processing Unit

C166S V2 CPU represents the third generation of the well known C166 core family. It combines many powerful enhancements with compatibility to the C166 family. The new architecture results in high CPU performance, fast and efficient access to different kinds of memories, and proficient peripheral units integration.

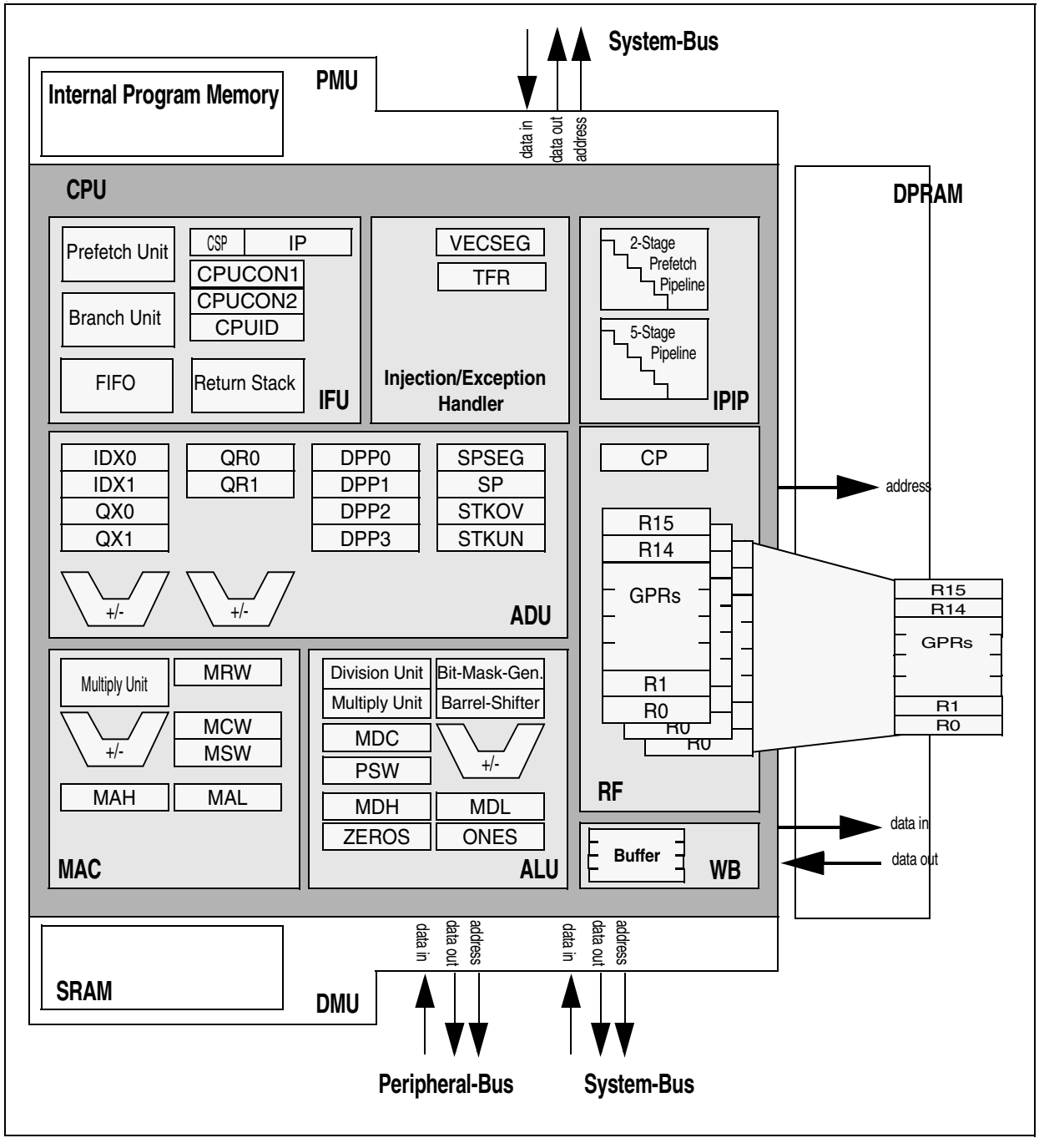

Figure 2-1 CPU Architecture

The new core architecture of the C166S V2 CPU results in higher CPU clock frequencies and reduces the number of clock cycles per executed instruction by half, compared to the C166 core. C166S V2 CPU also integrates a multiplication and accumulation unit which dramatically increases performance of the DSP-intensive tasks.

C166S V2 CPU has eight main units that are listed below. All of these units have been optimized to achieve maximum performance and flexibility.

- High Performance **I**nstruction **F**etch **U**nit (**IFU**)
  - High Bandwidth Fetch Interface
  - Instruction FIFO
  - High Performance Branch-, Call-, and Loop-Processing with instruction flow prediction
- Return Stack
  - Injection/Exception Handler
  - Handling of Interrupt Requests
  - Handling of Hardware Failures
- **I**nstruction **Pip**eline (**IPIP**)
  - Bypassable 2-stage Prefetch Pipeline
  - 5-stage Execution Pipeline
- **A**ddress and **D**ata **U**nit (**ADU**)
  - 16-bit arithmetic unit for address generation
  - DSP address unit with a set of dedicated address- and offset pointers
- **A**rithmetic and **L**ogic **U**nit (**ALU**)
  - 8-bit and 16-bit Arithmetic Unit
  - 16-bit Barrel Shifter
  - Multiplication and Division Unit
  - 8-bit and 16-bit Logic Unit
  - Bit manipulation Unit
- **M**ultiply and **AC**cumulate Unit (**MAC**)
  - 16-bit multiplier with 32-bit result generation[1)]
  - 40-bit Accumulator with 40-bit Barrel Shifter
  - Repeat Control Unit
- **R**egister **F**ile (**RF**)
  - 5-port Register File with three independent register banks
- **W**rite **B**ack **B**uffer (**WB**)
  - 3-entries buffer

1) The same hardware-multiplier is used in the ALU and in the MAC Unit.

2.1 Register Description Format

C166S V2 CPU contains a set of Special Function Register (SFR) and Extended Special Function Registers (ESFR). They are described in the respective chapter of this manual. The example below shows how to interpret the format and notation used to describe SFRs and ESFRs.

A word register looks like this:

REG_NAME
Short Description **SFR(b)/ESFR(b)/XSFR** **Reset Value: aaaa$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	bitfield A					0	0	bit C	bit B	bit A
r	r	r	r	r	r	rwh					r	r	rw	rw	rwh

A byte register looks like this:

REG_NAME
Short Description **SFR(b)/ESFR(b)/XSFR** **Reset Value: aa$_H$**

7	6	5	4	3	2	1	0
0	bitfield A			0	bit C	bit B	bit A
r	rwh			r	rw	rw	rwh

Field	Bits	Type	Description
bitfieldX	[m:n]	type	**Description** value Function off(Default) value Enable Function 1 ... ...
bitX	[n]	type	**Description** 0 Function off(Default) 1 Enable Function

Elements:

REG_NAME	Name of this register
bitX	Name of bit
bitfieldX	Name of bitfield
A16 / A8	Long 16-bit address/Short 8-bit address
SFR(b)/ESFR(b)	Register space (SFR or ESFR (bit addressable) Register)
XSFR	Register located in the internal 4 k IO area

(* *) * *	Register contents after reset	
	'0/1'	: defined value,
	'U'	: unchanged (undefined ('**X**') after power up)
	'?'	: defined by reset configuration
[n]	Bit number	
[m:n]	n	: Bit number first bit of the bitfield
	m	: Bit number of last bit of the bitfield
type	**'r'**	: readable by software
	'w'	: writable by software
	'h'	: writable by hardware
value	**'0/1'**	: defined value,
	'X'	: undefined,
	'0'	: reserved for future purpose, read access delivers 0, must not be set to 1

2.2 CPU Special Function Registers

The core CPU requires a set of CPU Special Function Registers (CSFRs) to maintain the system state information, to control system and bus configuration, and to manage code memory segmentation and data memory paging. The CPU also uses CSFRs to access the General Purpose Registers (GPRs) and the System Stack, to supply the ALU with register-addressable constants, and to support multiply and divide ALU operations.

The access mechanism for these CSFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can be controlled by any instruction capable of addressing the SFR/CSFR memory space, there is no need for special system control instructions.

However, to ensure proper processor operations, certain restrictions on the user access to some CSFRs must be imposed. For example, the Instruction Pointer (IP) and Code Segment Pointer (CSP) cannot be accessed directly at all. They can only be changed indirectly via branch instructions.

The PSW, SP, and MDC registers can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing.

Note: Note that any explicit write request (via software) to an CSFR supersedes a simultaneous modification by hardware of the same register.

Note: All SFRs may be accessed wordwise, or bytewise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. Any write operation to a single byte of an CSFR clears the non-addressed complementary byte within the specified CSFR.
Non-implemented (reserved) CSFR bits cannot be modified, and will always supply a read value of 0.

2.3 Instruction Fetch and Program Flow Control

The Instruction Fetch Unit (IFU) pre-fetches and pre-processes instructions to provide a continuous instruction flow. The IFU can fetch simultaneously at least two instructions via a 64-bit wide bus from the Program Management Unit (PMU). The pre-fetched instructions are stored in an instruction FIFO. Pre-processing of branch instructions enables the instruction flow to be predicted. While the CPU is in the process of executing an instruction fetched from the FIFO, the pre-fetcher of the IFU starts to fetch a new instruction at a predicted target address from the PMU. The latency time of this access is hidden by the execution of the instructions which have been buffered in the FIFO before. Even for a non-sequential instruction, execution the IFU can generally provide a continuous instruction flow. The IFU contains two pipeline stages: the Prefetch Stage and the Fetch Stage.

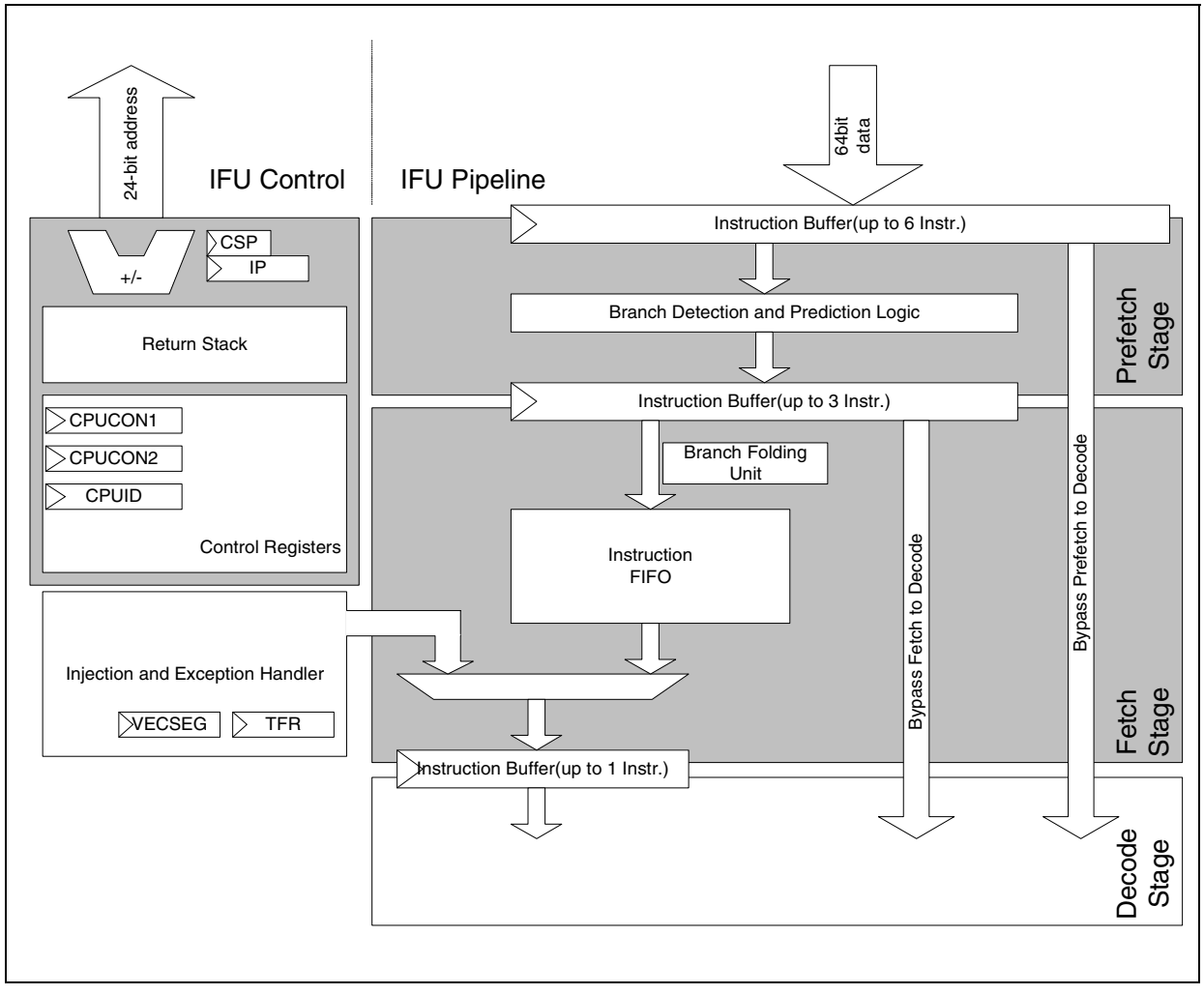

Figure 2-2 IFU Block Diagram

During the pre-fetch stage, the Branch Detection and Prediction Logic analyzes up to three pre-fetched instructions stored in the first Instruction Buffer (up to six instructions). If a branch is detected, then the IFU starts to fetch the next instructions from the PMU according to the prediction rules. After having been analyzed, up to three instructions are stored in the second Instruction Buffer (three instructions) which is the input register of the Fetch Stage.

On the Fetch Stage, the pre-fetched instructions are stored in the instruction FIFO. The Branch Folding Unit (BFU) allows processing of branch instructions in parallel with preceding instructions. To achieve this the BFU pre-processes and re-formats the branch instruction. First, BFU defines (calculates) the absolute target address. This address—after being combined with branch condition and branch attribute bits—is stored in the same FIFO step as the preceding instruction. The target address is also used to pre-fetch the next instructions.

For the Execution Pipeline, both instructions are fetched from the FIFO again and are executed in parallel. If the instruction flow was predicted incorrectly (or FIFO is empty), the two stages of the IFU can be bypassed.

Note: Pipeline behavior in case of a incorrectly predicted instruction flow is described in the following sections.

2.3.1 Branch Target Addressing Modes

The target address and the segment of jump or call instructions can be specified by several addressing modes. The Instruction Pointer register (IP) may be updated using relative, absolute, or indirect modes. The Code Segment Pointer register (CSP) can be updated using an absolute value only. A special mode is provided to address the interrupt and trap jump vector table which resides in the lowest portion of the code segment selected by the VECSEG register contents.

Table 2-1 Branch Target Addressing Modes

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000_H...$FFFE_H$
rel	(IP) = (IP) + 2*rel (IP) = (IP) + 2*(rel+1)	- -	rel = 00_H...$7F_H$ rel = 80_H...FF_H
[Rw]	(IP) = (Rw)	-	Rw w = 0...15
seg	-	(CSP) = seg	seg = 0...255(3)
#trap7	(IP) = 0000_H + VECSC*trap7	(CSP) = VECSEG	trap7 = 00_H...$7F_H$

caddr: Specifies an absolute 16-bit code address within the current segment. Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of 'caddr' is not used.

rel: This mnemonic represents an 8-bit signed word offset address relative to the current Instruction Pointer contents, which points to the instruction after the branch instruction. Depending on the offset address range, both forward ('rel'= 00_H to $7F_H$) and backward ('rel'= 80_H to FF_H) branches are possible. The branch instruction itself is repeatedly executed, when 'rel' = '-1' (FF_H) for a word-sized branch instruction, or 'rel' = '-2' (FE_H) for a double-word-sized branch instruction.

[Rw]: In this case, the 16-bit branch target instruction address is determined indirectly by the contents of a word GPR. In contrast to indirect data addresses, indirectly specified code addresses are **NOT** calculated via additional pointer registers (eg. DPP registers). Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of 'caddr' is not used.

seg: Specifies an absolute code segment number. The C166S V2 CPU supports 256 different code segments, so only the eight lower bits (respectively) of the 'seg' operand value are used to update the CSP register.

#trap7: Specifies a particular interrupt or trap number for branching to the corresponding interrupt or trap service routine via a jump vector table. Trap numbers from 00_H to $7F_H$ can be specified to access any double word code location within the address range xx'0000_H...xx'$15D4_H$ (depending of VECSC) in the selected code segment (see VECSEG, i.e. the interrupt jump vector table), please refer to **Section 5.1.4**.

2.3.2 Branch Detection and Branch Prediction

The Branch Detection Unit pre-processes instructions and classifies detected branches. Depending on the branch class, the Branch Prediction Unit predicts the program flow using the rules in the following table:.

Table 2-2 Branch Target Addressing Modes

Instruction Classes	Instructions	Prediction
Branch instructions with user programmable branch prediction	JMPA- xcc,caddr JMPA+ xcc,caddr CALLA- xcc, caddr CALLA+ xcc,caddr	The User can specify whether the branch should be taken
Branch instructions with branch prediction defined by Assembler	JMPA xcc,caddr CALLA xcc, caddr	Assembler defines whether the branch should be taken based on the jump condition.
Inter-segment branch instructions	JMPS seg, caddr CALLS seg,caddr	The branch is always taken.
Indirect branch instructions	JMPI cc,[Rw] CALLI cc,[Rw]	The branch is taken only if the branch is unconditional.
Relative branches instructions with condition code	JMPR cc,rel	The branch is taken if it is unconditional or if the branch is a backward branch.
Relative branch instructions without condition code	CALLR rel	The branch is always taken.
Branch instructions with bitcondition	JB bitaddr,rel JBC bitaddr,rel JNB bitaddr,rel JNBS bitaddr,rel	The branch is taken if it is a backward branch. Forward branches are always not taken.
Return instructions	RET RETS RETP RETI	The branch is always taken.

Note: For JMPA+/- and CALLA+/- instructions, a static user programmable prediction scheme is used. If bit 8 (’a’) of the instruction long word is cleared, the branch is assumed ‘taken.’ If it is set, the branch is assumed ‘not taken’. The user controls value of bit 8 by entering ’+’ or ’-’ in the instruction mnemonics. This bit can be also set/cleared by the Assembler for JMPA and CALLA instructions depending on the jump condition.

Note: For JMPA instruction, a pre-fetch hint bit is used (the instruction bit 9 = I). This bit is required by the fetch unit to deal efficiently with short backward loops. It must be set if 0 < IP_jmpa - IP_target <= 32, where IP_jmpa is the address of the JMPA instruction and IP_target is the target address of the JMPA. Otherwise, bit 9 must be cleared.

2.3.3 Sequential and Mispredicted Instruction Flow

Because passing through one pipeline stage takes at least one clock cycle, any isolated instruction takes at least five clock cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to five instructions (with branches up to six instructions). Therefore, most of the instructions appear to be processed during one clock cycle as soon as the pipeline has been filled once after reset.

The pipelining increases the average instruction throughput considered over a certain period of time. In this manual, any execution time specification always refers to the average instruction execution time due to pipelined parallel processing.

2.3.3.1 Correctly Predicted Instruction Flow

Figure 2-3 and **Figure 2-4** show the continuous execution of instructions in principal under the assumption of a fast (0 wait states) Program Memory. In this example, most of the instructions are executed in one CPU cycle while Instruction I_{n+6} takes two CPU cycles for the execution. I_{n+6} is a general example for multicycle instructions (two cycles instruction in this case).

The instructions are fetched from the Instruction FIFO while the IFU pre-fetches the next instructions to fill the FIFO. The Instruction FIFO is being filled with new instructions while the previously stored instructions are being fetched from the FIFO to be executed in the CPU. As long as the instruction flow is correctly predicted by the IFU, both processes are independent.

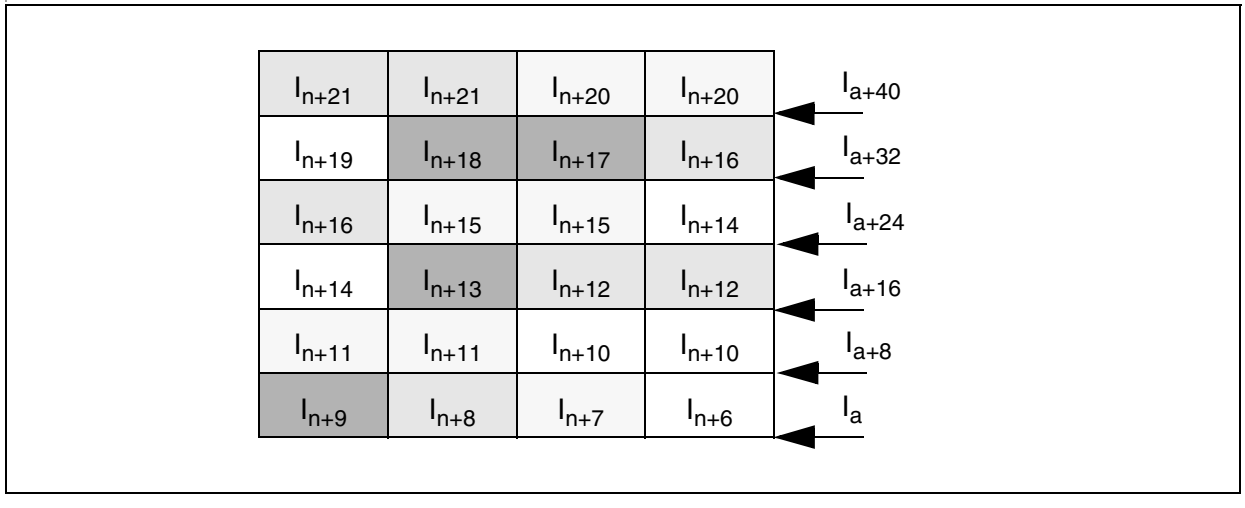

Figure 2-3 Program Memory Contents for Figure 2-4

The diagram shows the sequential instruction flow through the different pipeline stages. While the Prefetcher is prefetching the instruction from the PMU, the processing pipeline is filled with instructions fetched out of the FIFO. In this example with a fast Internal Program Memory, the Prefetcher is able to fetch more instructions than the processing pipeline can execute. In T_{n+4}, the FIFO and prefetch buffer are filled and no further

instructions can be prefetched. The PMU address stays stable (T_{n+4}) until a whole 64-bit double word can be buffered (T_{n+7}) in the 96-bit Prefetch buffer again.

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}	T_{n+8}
PMU Address	I_{a+16}	I_{a+24}	I_{a+32}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+48}	I_{a+48}
PMU Data 64bit	I_{d+1}	I_{d+2}	I_{d+3}	I_{d+4}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+7}
PREFETCH 96 bit Buffer	I_{n+6} … I_{n+9}	I_{n+9} … I_{n+11}	I_{n+12} I_{n+13}	I_{n+14} I_{n+15}	I_{n+15} … I_{n+19}	I_{n+15} … I_{n+19}	I_{n+16} … I_{n+19}	I_{n+17} … I_{n+19}	I_{n+18} … I_{n+21}
FETCH Instruction Buffer	I_{n+5}	I_{n+6} I_{n+7} I_{n+8}	I_{n+9} I_{n+10} I_{n+11}	I_{n+12} I_{n+13}	I_{n+14}	-	I_{n+15}	I_{n+16}	I_{n+17}
FIFO contents	I_{n+3} … I_{n+5}	I_{n+4} … I_{n+8}	I_{n+5} … I_{n+11}	I_{n+6} … I_{n+13}	I_{n+7} … I_{n+14}	I_{n+7} … I_{n+14}	I_{n+8} … I_{n+15}	I_{n+9} … I_{n+16}	I_{n+10} … I_{n+17}
Fetch from FIFO	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+7}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}	I_{n+11}
DECODE	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}
ADDRESS	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}
MEMORY	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}
EXECUTE	I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}
WRITE BACK		I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}

Figure 2-4 Sequential Instruction Execution