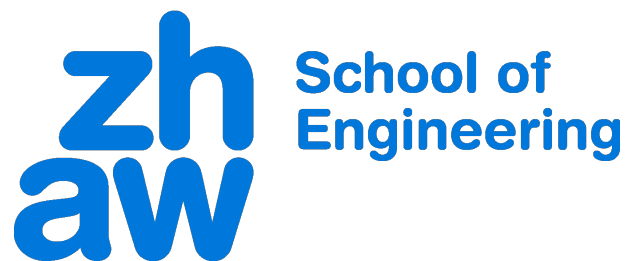


ZHAW Zurich University of Applied Sciences
Winterthur

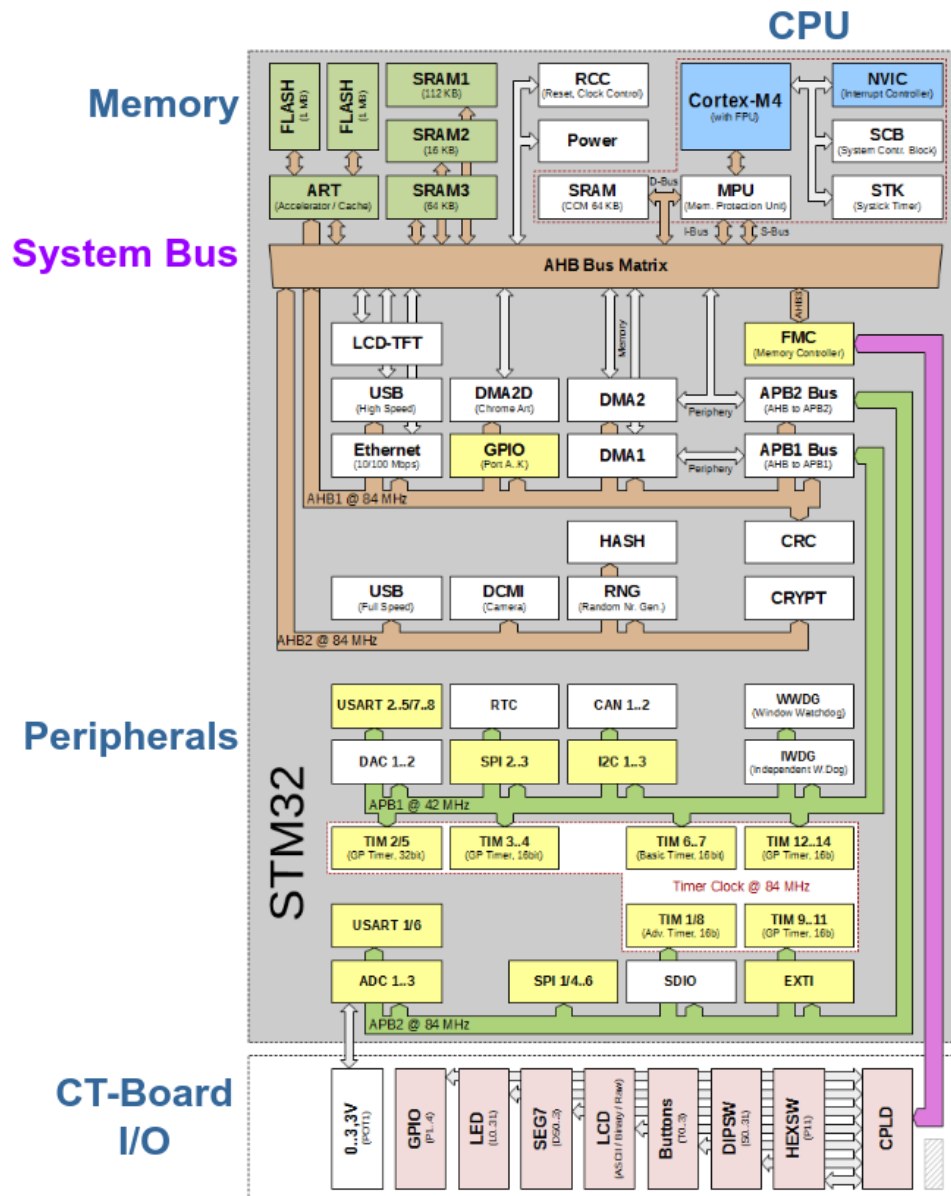


Zusammenfassung CT2

Studienwochen 1-10

Written by: Severin Sprenger
June 17, 2026
Zf. CT2 SW 1-10

1 General



Bus Timing Options

- Synchronous**
 - Master and slaves use a common clock¹⁾
 - Clock edges control bus transfer on both sides
 - Used by most on-chip busses
 - Off-chip: DDR and synchronous RAM
- Asynchronous**
 - Slaves have no access to the clock of the master
 - Control signals carry timing information to allow synchronization
 - Widely used for low data-rate off-chip memories
→ parallel flash memories and asynchronous RAM

Access through Pointers

- E.g. writing to and reading from CT Board I/O

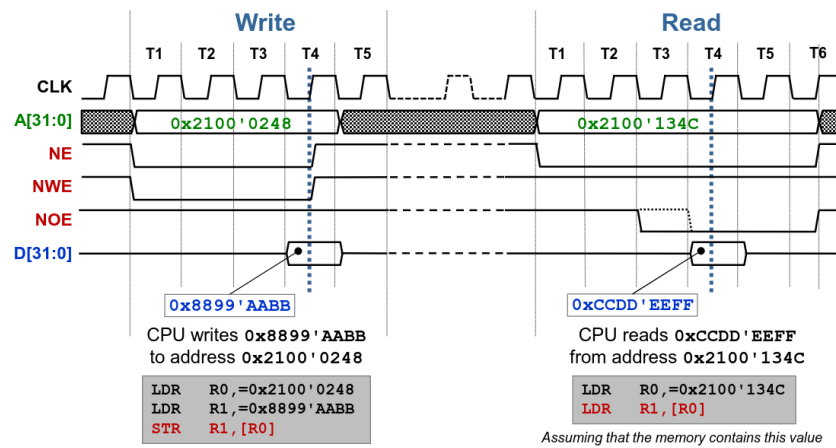
```
// a pointer called p_reg pointing to
// a volatile uint32_t
volatile uint32_t *p_reg;

// set LEDs
p_reg = (volatile uint32_t *) (0x60000100);
*p_reg = 0xAA55AA55;

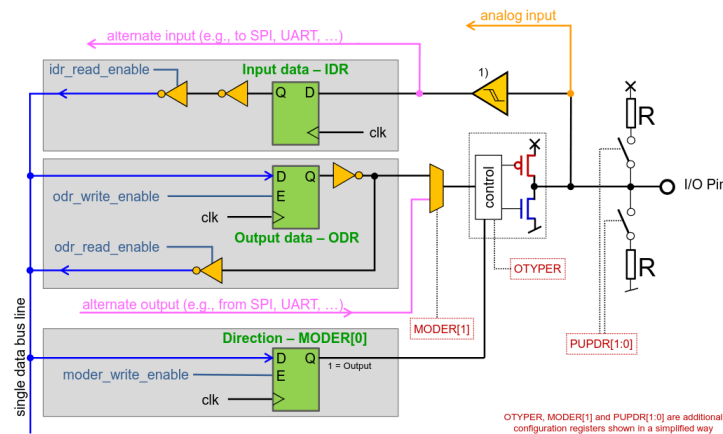
// wait for dip_switches to be non-zero
p_reg = (volatile uint32_t *) (0x60000200);
while ( *p_reg == 0 ) { }
```

Annotations in the code:

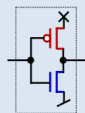
- cast 'unsigned integer' to 'pointer to volatile uint32_t'
- write 0xAA55'AA55 to LEDs
- read dip-switches



2 GPIO



OT = '0' → push-pull



Output stage can drive output 'high' or 'low'

OT = '1' → open-drain

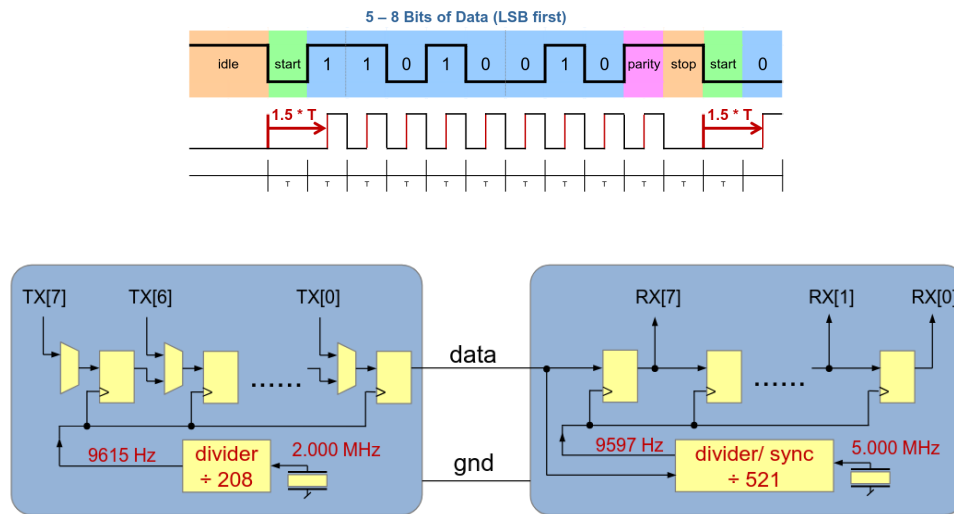


Output stage can only drive output 'low'

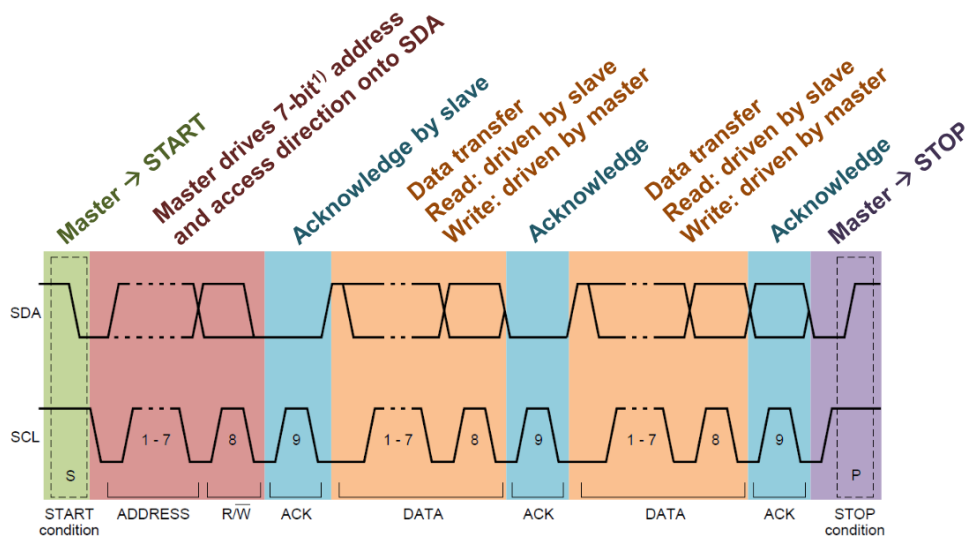
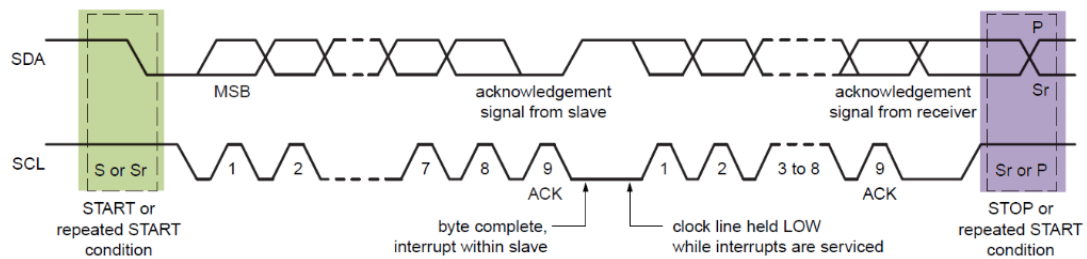
3 Peripheral Buses

UART	SPI	I2C
serial ports (RS-232)	4-wire bus	2-wire bus
TX, RX opt. control signals	MOSI, MISO, SCLK, SS	SCL, SDA
point-to-point	point-to-multipoint	(multi-) point-to-multi-point
full-duplex	full-duplex	half-duplex
asynchronous	synchronous	synchronous
only higher layer addressing	slave selection through $\overline{SS}$ signal	7/10-bit slave address
parity bit possible	no error detection	no error detection
chip-to-chip, PC terminal program	chip-to-chip, on-board connections	chip-to-chip, board-to-board connections
The three interfaces provide the lowest layer of communication and require higher level protocols to provide and interpret the transferred data.		

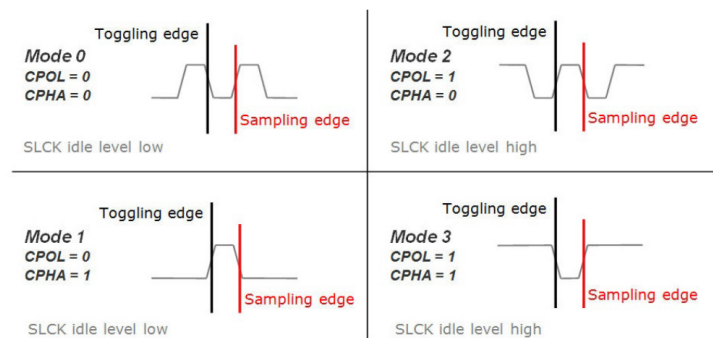
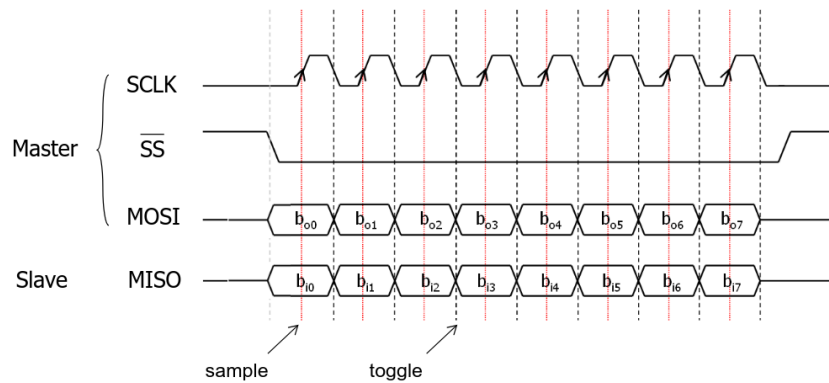
3.1 UART



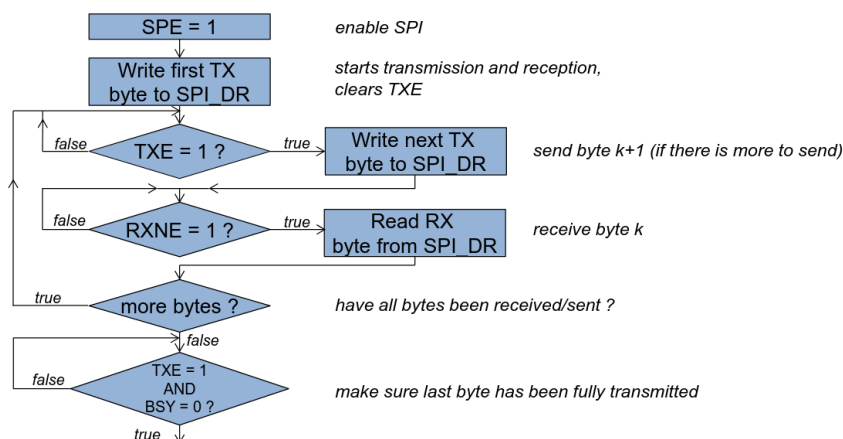
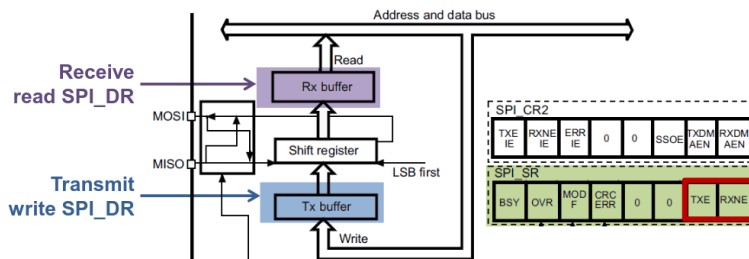
3.2 I2C



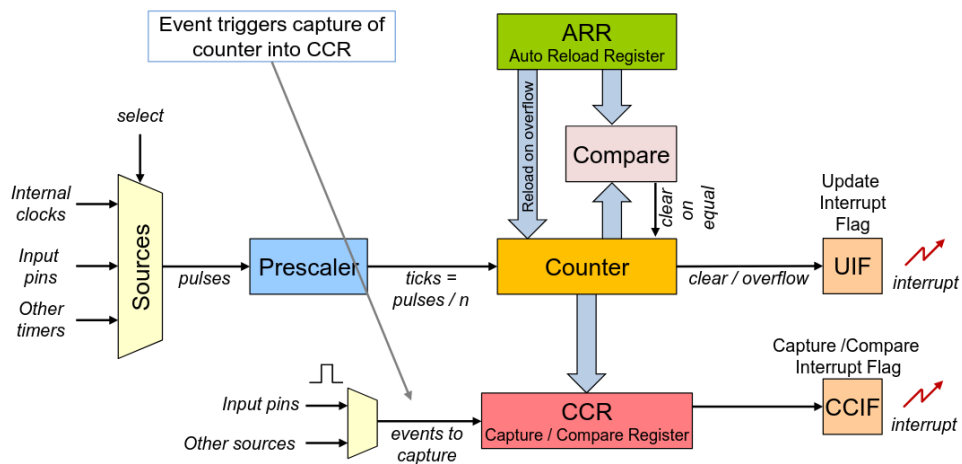
3.3 SPI



- TXE TX Buffer Empty Software can write next TX Byte to register SPI_DR
- RXNE RX Buffer Not Empty A byte has been received. Software can read it from SPI_DR

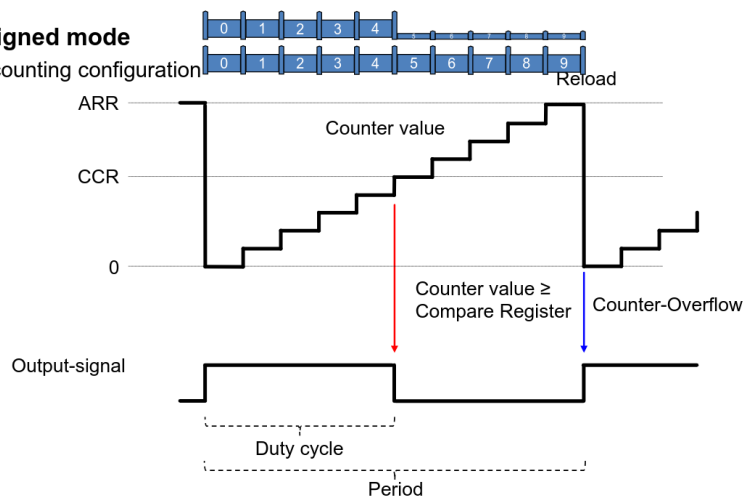


4 Timer



Edge-aligned mode

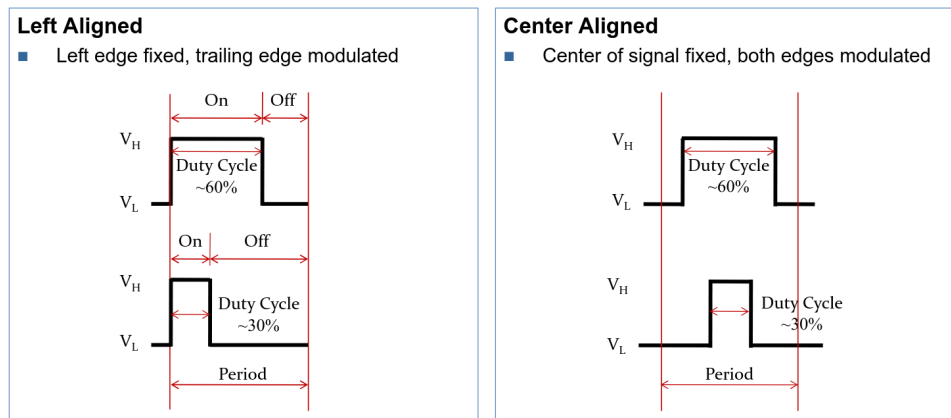
- Up-counting configuration



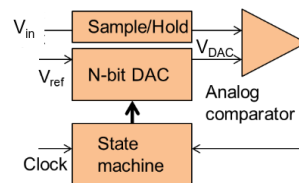
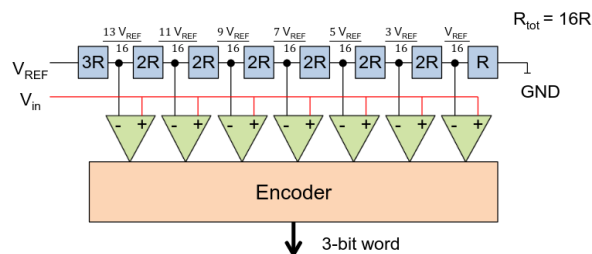
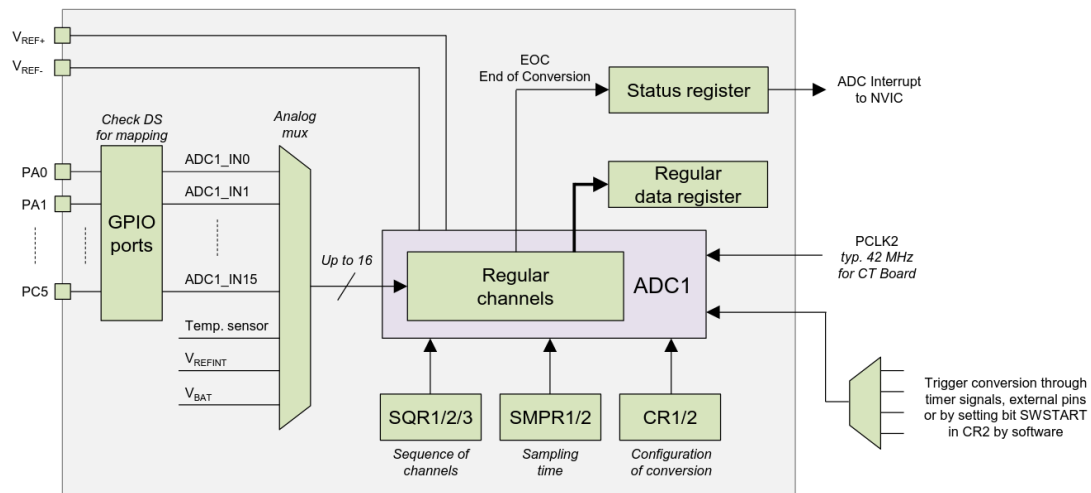
4.1 PWM

$$\text{Duty Cycle} = \frac{\text{On Time}}{\text{Period}} \cdot 100 \%$$

$$V_{avg} = D \cdot V_N + (1 - D) \cdot V_L$$



5 ADC

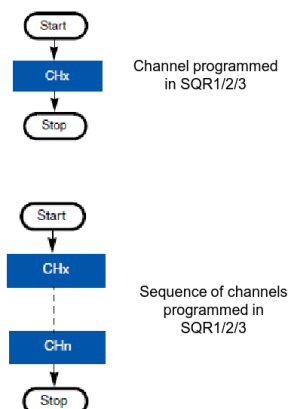


■ Single channel single conversion

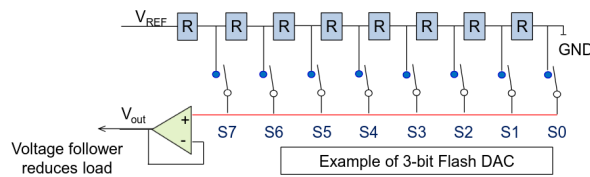
- Possible use
 - Sensor tied to ADC input
 - On a push button you read and display it

■ Multi-channel single conversion

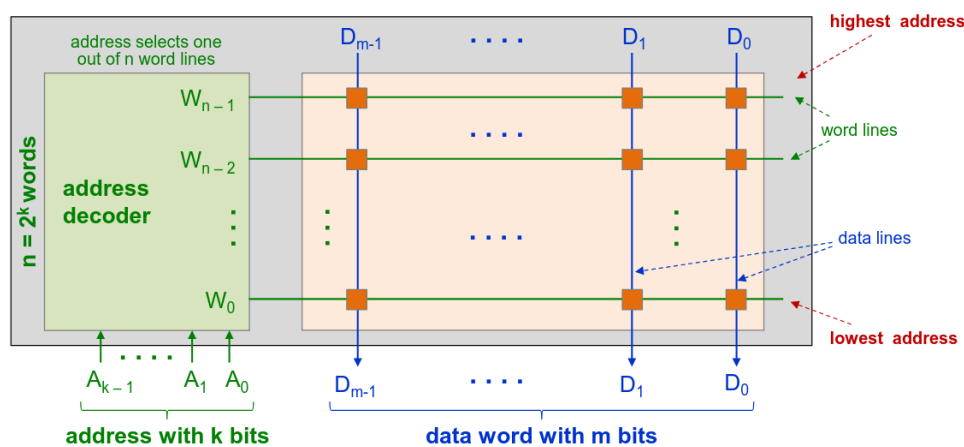
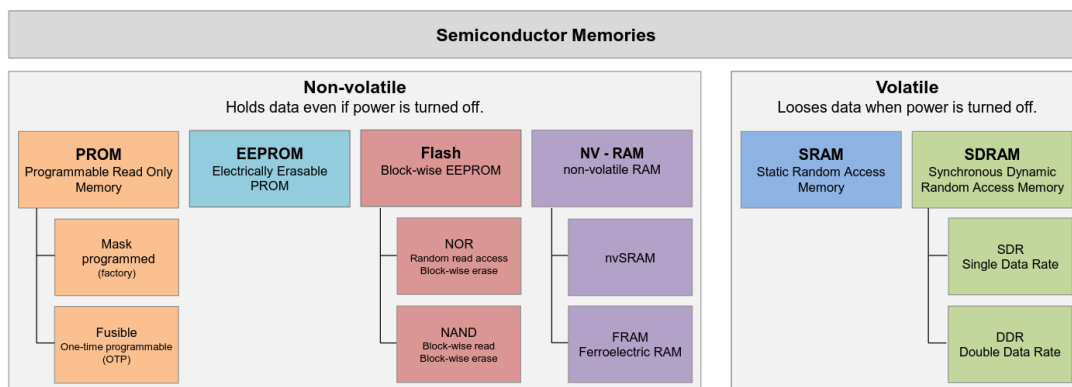
- ADC sequencer used to set up order of conversions.
- Possible use
 - Group of sensors need to be monitored
 - For each sensor there is an ADC input
 - Sensors are grouped and scanned one after the other every time a reading needs to be done



6 DAC



7 Memory



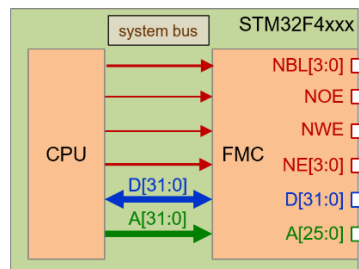
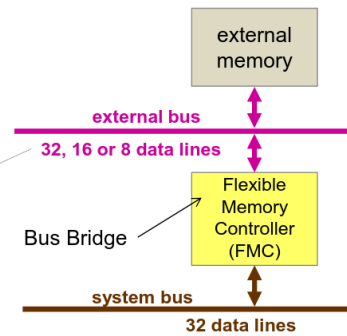
	NOR Flash	NAND Flash
Topology		
Applications	- Execute code directly from memory - Persistent device configurations (replacement of EEPROM)	- File-based IO, disks - Large amounts of sequential data (images, SD cards, SSD) - Load programs into RAM before executing
Density	Medium Up to 2 GBit = 256 MByte	High Up to 1 Tbit = 128 GByte
Interface	- Read same as asynchronous SRAM - Types with serial interface (SPI) available	- Special NAND flash interface - Error correction for defective blocks
Access	- Random access read ~0.12 μs - Writing individual bytes possible - Slow writes ~180 μs / 32 Byte	- Slow random access read: 1. Byte 25 μs, then 0.03 μs each - Writing of individual bytes difficult - Fast block write ~300 μs / 2¹¹² Bytes

7.1 FMC

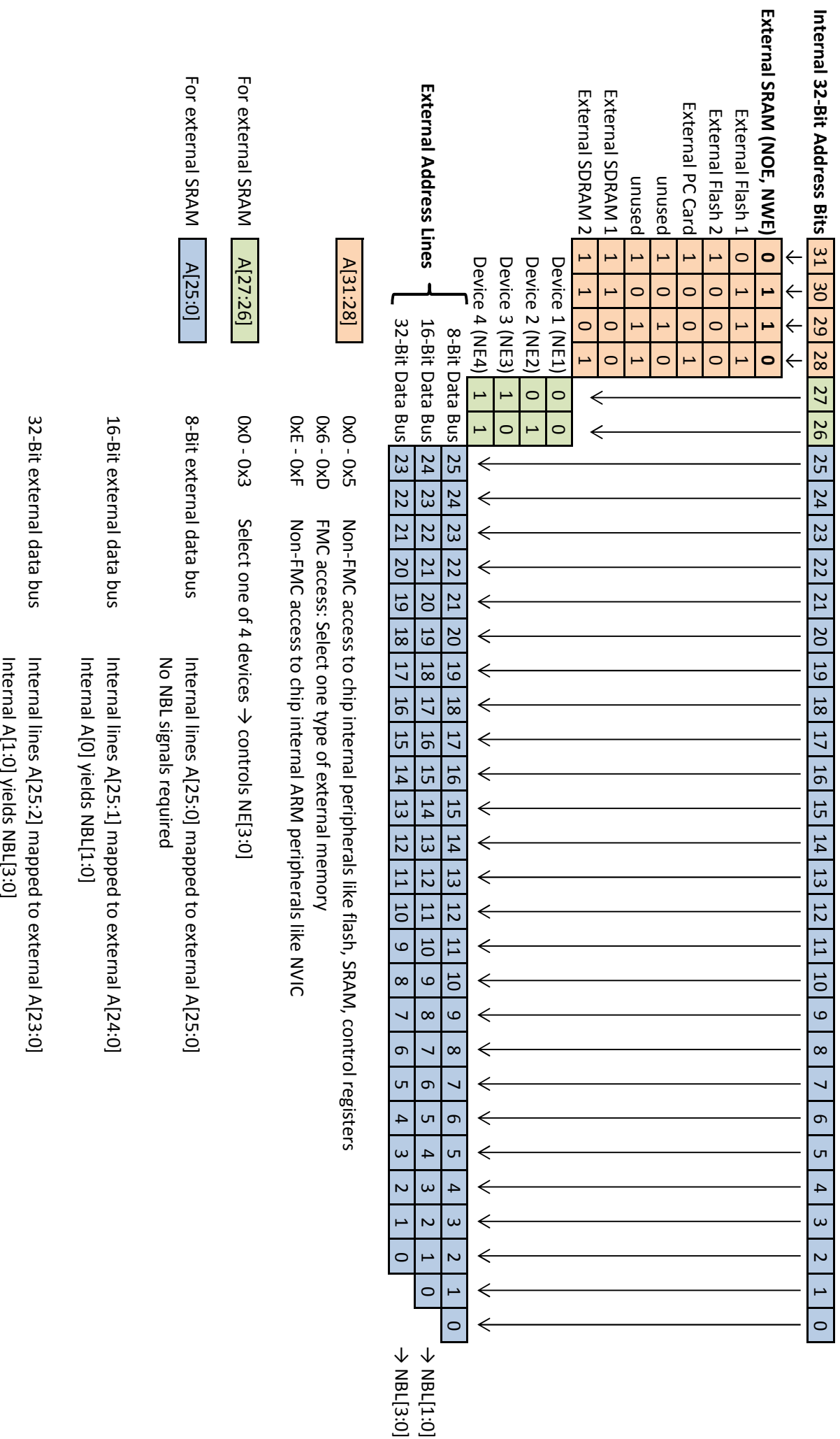
- Bridge between system bus and external bus
 - Slave on system bus
 - Master on external bus

- System bus accesses
 - In address range 0x6000'0000 to 0xDFFF'FFFF
 - Bridged to external bus
 - I.e. FMC initiates a bus cycle on external bus

Number of data lines is a design decision and depends on the external memory device



STM32F429 Flexible Memory Controller (FMC) Decoding



Datenblattauszug GPIO

Boundary address	Peripheral	Bus	Register map
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	Section 35.12.6: OTG_HS register map on page 1445
0x4002 B000 - 0x4002 BBFF	DMA2D		Section 11.5: DMA2D registers on page 349
0x4002 9000 - 0x4002 93FF	ETHERNET MAC		Section 33.8.5: Ethernet register maps on page 1214
0x4002 8C00 - 0x4002 8FFF			
0x4002 8800 - 0x4002 8BFF			
0x4002 8400 - 0x4002 87FF			
0x4002 8000 - 0x4002 83FF			
0x4002 6400 - 0x4002 67FF	DMA2		Section 10.5.11: DMA register map on page 332
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 4000 - 0x4002 4FFF	BKPSRAM		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		Section 3.9: Flash interface registers
0x4002 3800 - 0x4002 3BFF	RCC		Section 7.3.25: RCC register map on page 263
0x4002 3000 - 0x4002 33FF	CRC		Section 4.4.4: CRC register map on page 114
0x4002 2800 - 0x4002 2BFF	GPIOK		Section 8.4.11: GPIO register map on page 284
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		Section 8.4.11: GPIO register map on page 284
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		
0x4001 6800 - 0x4001 6BFF	LCD-TFT	APB2	Section 16.7.26: LTDC register map on page 504
0x4001 5800 - 0x4001 5BFF	SAI1		Section 29.17.9: SAI register map on page 944
0x4001 5400 - 0x4001 57FF	SPI6	APB2	Section 28.5.10: SPI register map on page 906
0x4001 5000 - 0x4001 53FF	SPI5		

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 2y:2y+1 MODERy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 OTy: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..I/J/K)

Address offset: 0x08

Reset values:

- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]	OSPEEDR14 [1:0]	OSPEEDR13 [1:0]	OSPEEDR12 [1:0]	OSPEEDR11 [1:0]	OSPEEDR10 [1:0]	OSPEEDR9 [1:0]	OSPEEDR8 [1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]	OSPEEDR6[1:0]	OSPEEDR5[1:0]	OSPEEDR4[1:0]	OSPEEDR3[1:0]	OSPEEDR2[1:0]	OSPEEDR1 [1:0]	OSPEEDR0 [1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 OSPEEDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: Fast speed

11: High speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 PUPDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

13.13.14 ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 48](#) and [Figure 49](#).

13.13.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **OVR**: Overrun

This bit is set by hardware when data are lost (either in single mode or in dual/triple mode). It is cleared by software. Overrun detection is enabled only when DMA = 1 or EOCS = 1.

0: No overrun occurred
1: Overrun has occurred

Bit 4 **STRT**: Regular channel start flag

This bit is set by hardware when regular channel conversion starts. It is cleared by software.

0: No regular channel conversion started
1: Regular channel conversion has started

Bit 3 **JSTRT**: Injected channel start flag

This bit is set by hardware when injected group conversion starts. It is cleared by software.

0: No injected group conversion started
1: Injected group conversion has started

Bit 2 **JEOC**: Injected channel end of conversion

This bit is set by hardware at the end of the conversion of all injected channels in the group. It is cleared by software.

0: Conversion is not complete
1: Conversion complete

Bit 1 **EOC**: Regular channel end of conversion

This bit is set by hardware at the end of the conversion of a regular group of channels. It is cleared by software or by reading the ADC_DR register.

0: Conversion not complete (EOCS=0), or sequence of conversions not complete (EOCS=1)
1: Conversion complete (EOCS=0), or sequence of conversions complete (EOCS=1)

Bit 0 **AWD**: Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in the ADC_LTR and ADC_HTR registers. It is cleared by software.

0: No analog watchdog event occurred
1: Analog watchdog event occurred

13.4 Data alignment

The ALIGN bit in the ADC_CR2 register selects the alignment of the data stored after conversion. Data can be right- or left-aligned as shown in [Figure 48](#) and [Figure 49](#).

The converted data value from the injected group of channels is decreased by the user-defined offset written in the ADC_JOFRx registers so the result can be a negative value. The SEXT bit represents the extended sign value.

For channels in a regular group, no offset is subtracted so only twelve bits are significant.

Figure 48. Right alignment of 12-bit data

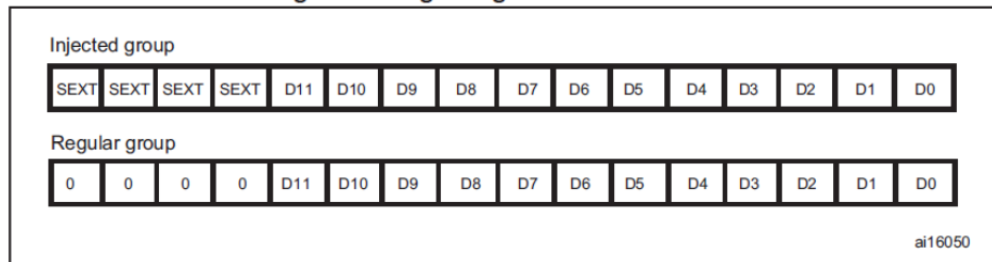
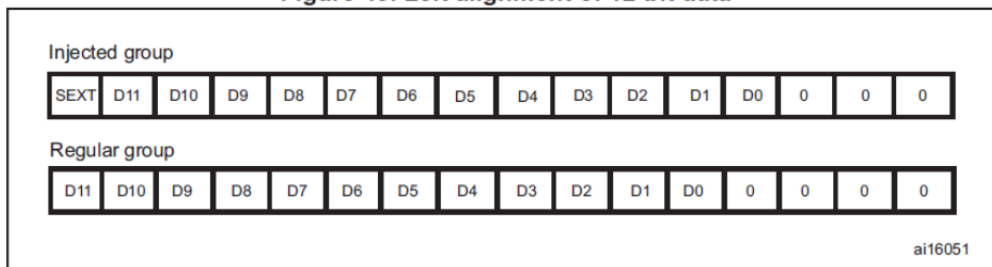


Figure 49. Left alignment of 12-bit data



Special case: when left-aligned, the data are aligned on a half-word basis except when the resolution is set to 6-bit. In that case, the data are aligned on a byte basis as shown in [Figure 50](#).

ADC Base Address			Address of Register
0x4001 2000	ADC1	Specific registers	0x4001 2000 + 0x000 + register offset
	ADC2	Specific registers	0x4001 2000 + 0x100 + register offset
	ADC3	Specific registers	0x4001 2000 + 0x200 + register offset
	Common	Common registers	0x4001 2000 + 0x300 + register offset

C Reference Card (ANSI)

Program Structure/Functions

<i>type func</i> (<i>type1</i> , ...);	function prototype
<i>type name</i> ;	variable declaration
int main(void) {	main routine
<i>declarations</i>	local variable declarations
<i>statements</i>	
}	
<i>type func</i> (<i>arg1</i> , ...) {	function definition
<i>declarations</i>	local variable declarations
<i>statements</i>	
return <i>value</i> ;	
}	
/* */	comments
int main(int argc, char *argv[])	main with args
exit(<i>arg</i>);	terminate execution

C Preprocessor

include library file	#include < <i>filename</i> >
include user file	#include " <i>filename</i> "
replacement text	#define <i>name</i> (<i>var</i>) <i>text</i>
replacement macro	Example. #define max(A,B) ((A)>(B) ? (A) : (B))
undefine	#undef <i>name</i>
quoted string in replace	#
Example. #define msg(A) printf("%s = %d", #A, (A))	
concatenate args and rescan	##
conditional execution	#if, #else, #elif, #endif
is <i>name</i> defined, not defined?	#ifdef, #ifndef
<i>name</i> defined?	defined(<i>name</i>)
line continuation char	\

Data Types/Declarations

character (1 byte)	char
integer	int
real number (single, double precision)	float, double
short (16 bit integer)	short
long (32 bit integer)	long
double long (64 bit integer)	long long
positive or negative	signed
non-negative modulo 2^m	unsigned
pointer to int, float,...	int*, float*,...
enumeration constant	enum <i>tag</i> { <i>name1=value1</i> ,...};
constant (read-only) value	<i>type</i> const <i>name</i> ;
declare external variable	extern
internal to source file	static
local persistent between calls	static
no value	void
structure	struct <i>tag</i> {...};
create new name for data type	typedef <i>type name</i> ;
size of an object (type is size_t)	sizeof <i>object</i>
size of a data type (type is size_t)	sizeof(<i>type</i>)

Initialization

initialize variable	<i>type name=value</i> ;
initialize array	<i>type name</i> []={ <i>value1</i> ,...};
initialize char string	char <i>name</i> []="string";

Constants

suffix: long, unsigned, float	65536L, -1U, 3.0F
exponential form	4.2e1
prefix: octal, hexadecimal	0, 0x or 0X
Example. 031 is 25, 0x31 is 49 decimal	
character constant (char, octal, hex)	'a', '\ooo', '\xhh'
newline, cr, tab, backspace	\n, \r, \t, \b
special characters	\\, \?, \', \"
string constant (ends with '\0')	"abc...de"

Pointers, Arrays & Structures

declare pointer to <i>type</i>	<i>type</i> * <i>name</i> ;
declare function returning pointer to <i>type</i> <i>*f</i> ();	
declare pointer to function returning <i>type</i> (<i>*pf</i> ());	
generic pointer type	void *
null pointer constant	NULL
object pointed to by <i>pointer</i>	* <i>pointer</i>
address of object <i>name</i>	& <i>name</i>
array	<i>name</i> [<i>dim1</i>][<i>dim2</i>]...
multi-dim array	
Structures	
struct <i>tag</i> {	structure template
<i>declarations</i>	declaration of members
};	

create structure	struct <i>tag name</i>
member of structure from template	<i>name.member</i>
member of pointed-to structure	<i>pointer -> member</i>
Example. (*p).x and p->x are the same	
single object, multiple possible types	union
bit field with <i>b</i> bits	unsigned member: <i>b</i> ;

Operators (grouped by precedence)

struct member operator	<i>name.member</i>
struct member through pointer	<i>pointer->member</i>
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, -, !, ~
indirection via pointer, address of object	* <i>pointer</i> , & <i>name</i>
cast expression to type	(<i>type</i>) <i>expr</i>
size of an object	sizeof
multiply, divide, modulus (remainder)	*, /, %
add, subtract	+, -
left, right shift [bit ops]	<<, >>
relational comparisons	>, >=, <, <=
equality comparisons	==, !=
and [bit op]	&
exclusive or [bit op]	^
or (inclusive) [bit op]	
logical and	&&
logical or	
conditional expression	<i>expr1</i> ? <i>expr2</i> : <i>expr3</i>
assignment operators	+=, -=, *=, ...
expression evaluation separator	,
Unary operators, conditional expression and assignment operators group right to left; all others group left to right.	

Flow of Control

statement terminator	;
block delimiters	{ }
exit from switch, while, do, for	break;
next iteration of while, do, for	continue;
go to	goto <i>label</i> ;
label	<i>label</i> : statement
return value from function	return <i>expr</i>
Flow Constructions	
if statement	if (<i>expr1</i>) <i>statement1</i> else if (<i>expr2</i>) <i>statement2</i> else <i>statement3</i>
while statement	while (<i>expr</i>) <i>statement</i>
for statement	for (<i>expr1</i> ; <i>expr2</i> ; <i>expr3</i>) <i>statement</i>
do statement	do <i>statement</i> while(<i>expr</i>);
switch statement	switch (<i>expr</i>) { case <i>const1</i> : <i>statement1</i> break; case <i>const2</i> : <i>statement2</i> break; default: <i>statement</i> }

ANSI Standard Libraries

<assert.h>	<ctype.h>	<errno.h>	<float.h>	<limits.h>
<locale.h>	<math.h>	<setjmp.h>	<signal.h>	<stdlib.h>
<stddef.h>	<stdio.h>	<stdlib.h>	<string.h>	<time.h>

Character Class Tests <ctype.h>

alphanumeric?	isalnum(c)
alphabetic?	isalpha(c)
control character?	isctrl(c)
decimal digit?	isdigit(c)
printing character (not incl space)?	isgraph(c)
lower case letter?	islower(c)
printing character (incl space)?	isprint(c)
printing char except space, letter, digit?	ispunct(c)
space, formfeed, newline, cr, tab, vtab?	isspace(c)
upper case letter?	isupper(c)
hexadecimal digit?	isxdigit(c)
convert to lower case	tolower(c)
convert to upper case	toupper(c)

String Operations <string.h>

s is a string; cs, ct are constant strings

length of s	strlen(s)
copy ct to s	strcpy(s,ct)
concatenate ct after s	strcat(s,ct)
compare cs to ct	strcmp(cs,ct)
only first n chars	strncmp(cs,ct,n)
pointer to first c in cs	strchr(cs,c)
pointer to last c in cs	strrchr(cs,c)
copy n chars from ct to s	memcpy(s,ct,n)
copy n chars from ct to s (may overlap)	memmove(s,ct,n)
compare n chars of cs with ct	memcmp(cs,ct,n)
pointer to first c in first n chars of cs	memchr(cs,c,n)
put c into first n chars of s	memset(s,c,n)

