KNSW

Wydział Elektroniki Politechniki Wrocławskiej

Warsztaty programowania układów mikroprocesorowych STM32

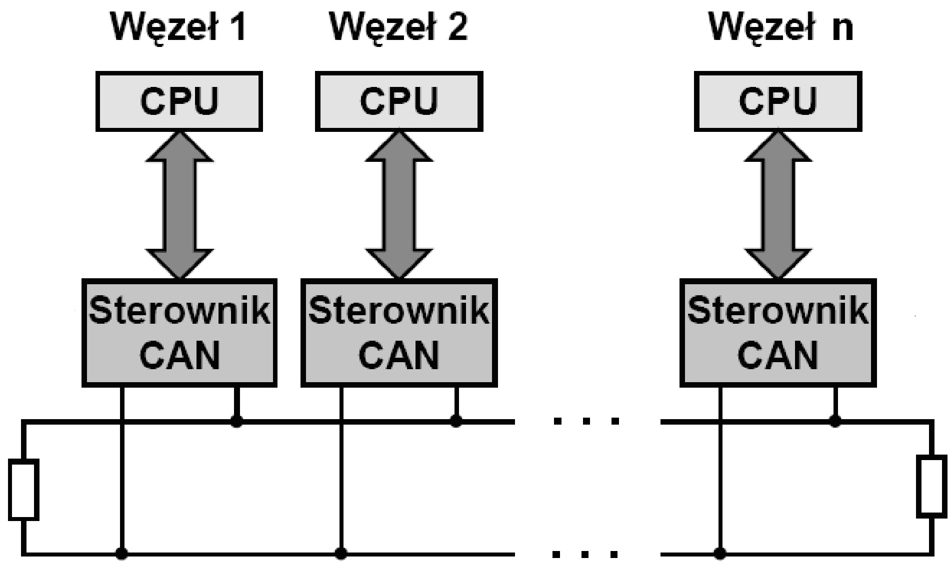
4. CAN (ang. Controller Area Network)

Szymon Panecki

Koło Naukowe Systemów Wbudowanych

Charakterystyka magistrali CAN

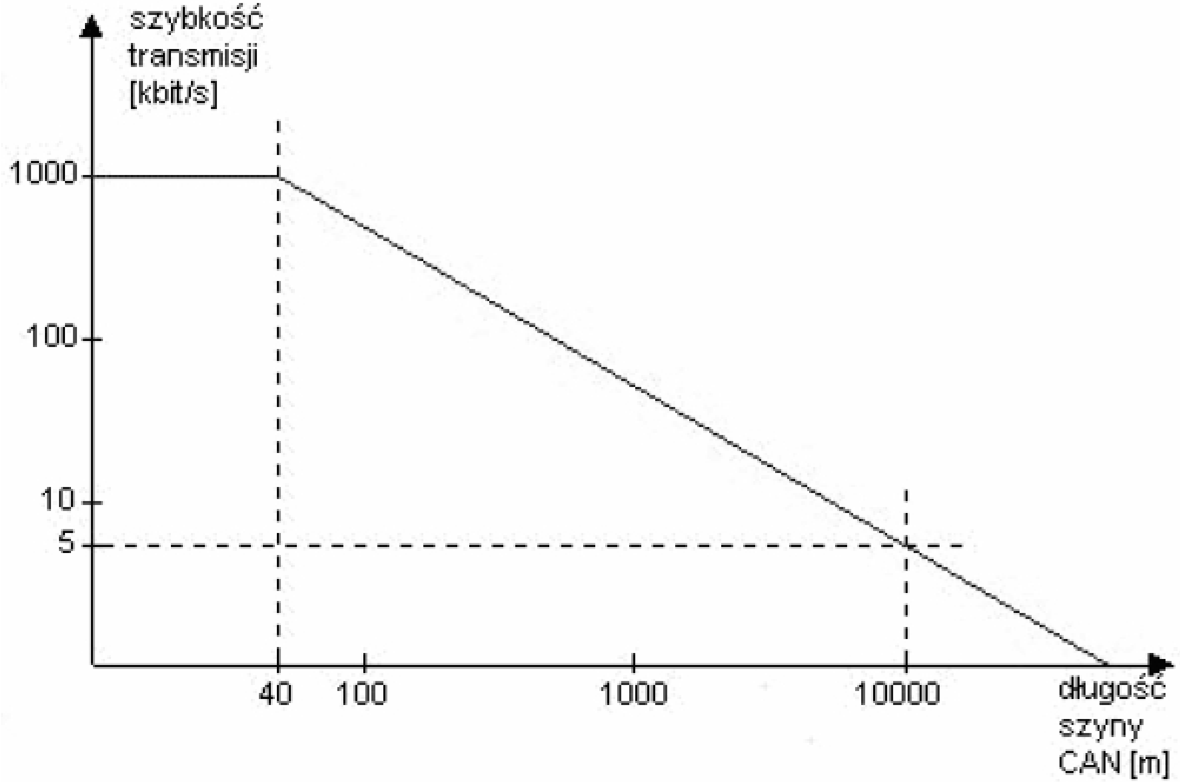
CAN (Controller Area Network) - szeregowa magistrala komunikacyjna, która powstała w roku 1989 w firmie Bosch z myślą o zastosowaniach w przemyśle samochodowym dla sterowania układami pomiarowymi i wykonawczymi.



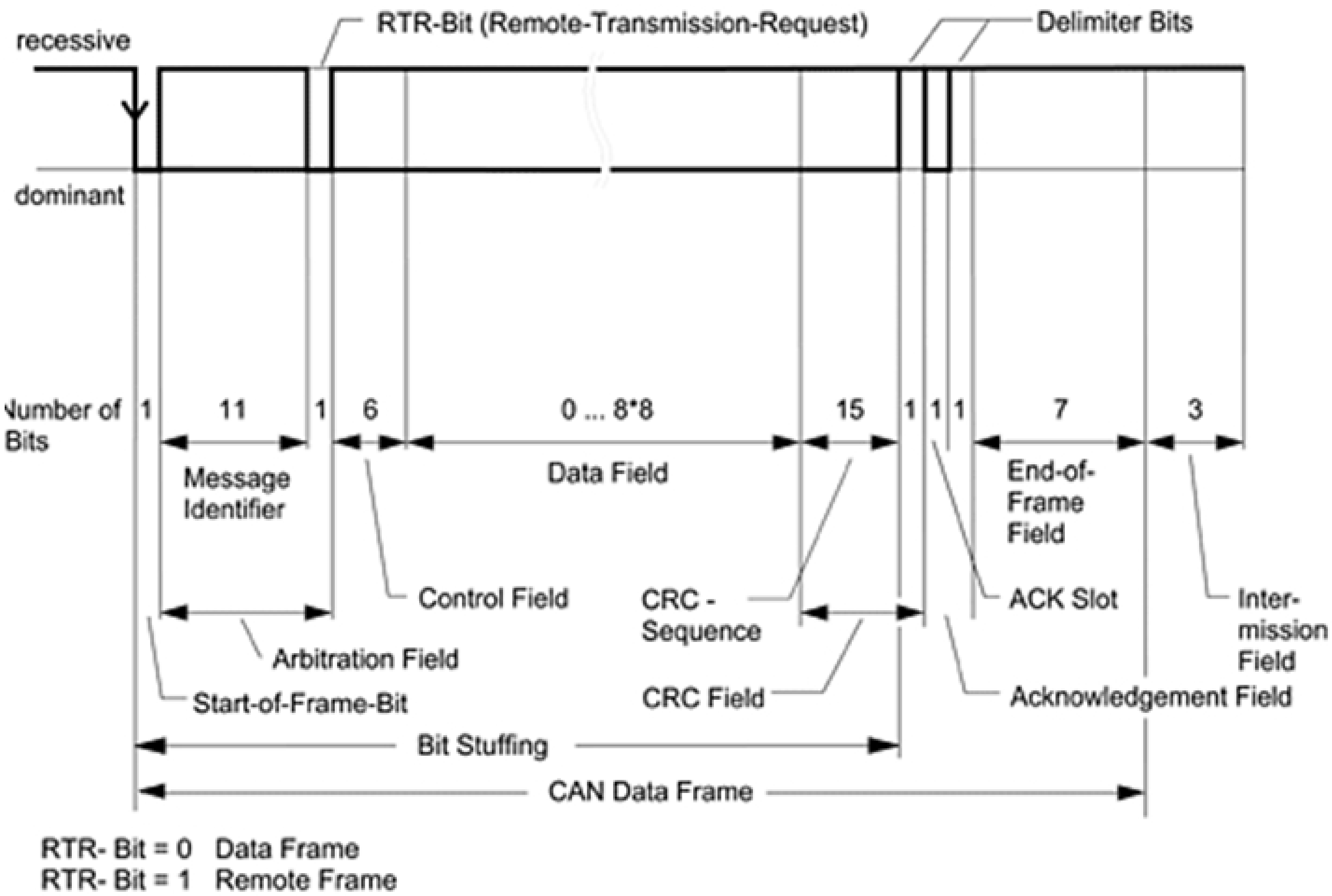
Cechy:

- Niezawodność
- Duża szybkość transmisji
- Duża odległość, na którą można przesyłać informacje
- Kontrola wykrywania błędów
- Odporność na zakłócenia

Zależność prędkości transmisji od długości magistrali

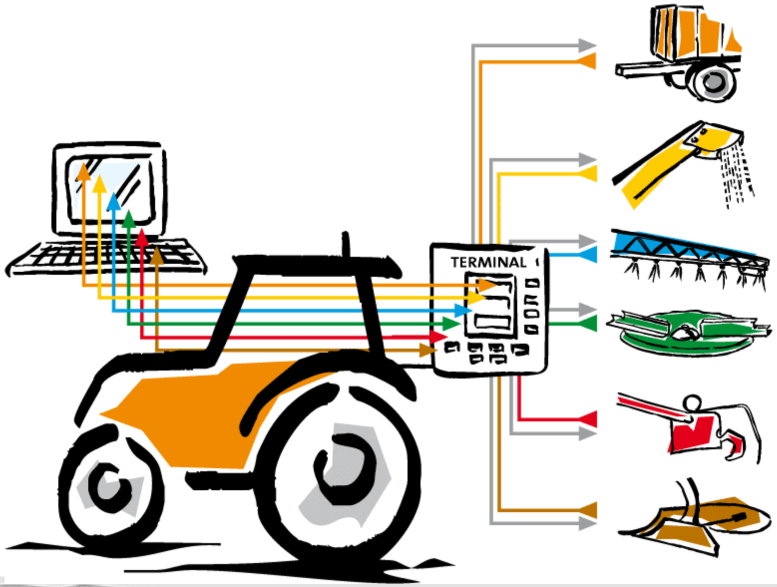


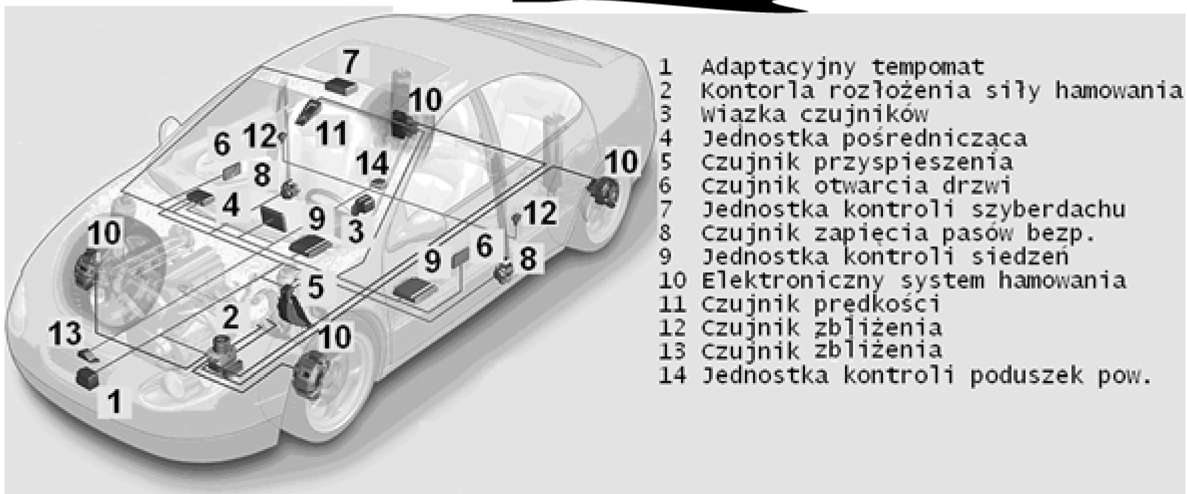
Budowa ramki CAN 2.0A



Zastosowanie

- Motoryzacja (J1939, ISOBUS)
- Sieci przemysłowe (DeviceNet)
- Statki (NMEA-2000)
- Kosmonautyka (satelita SSETI ESEO)
- Wiele innych

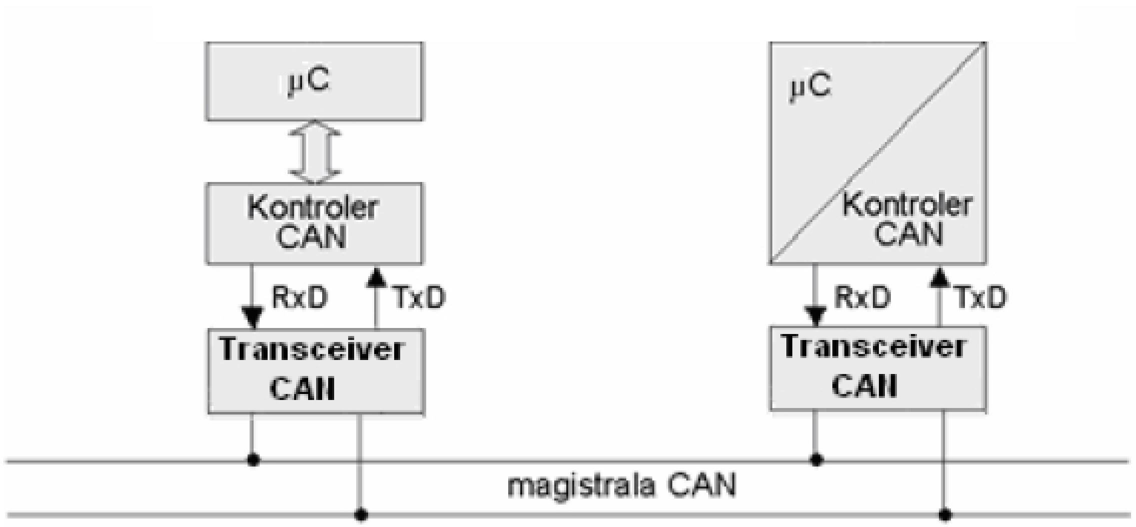




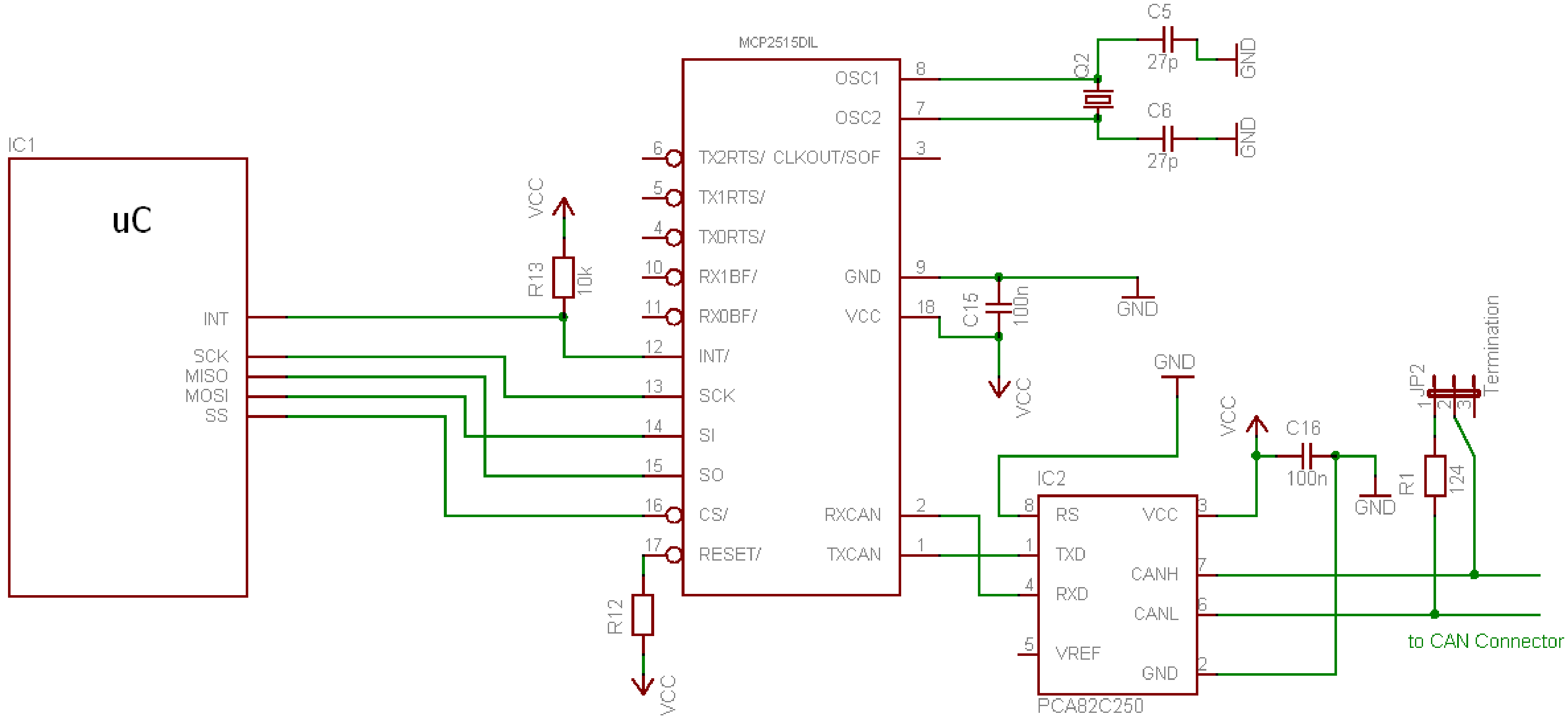
Budowa węzła CAN

- Układ mikroprocesorowy z wbudowanym kontrolerm transmisji CAN , transceiver CAN

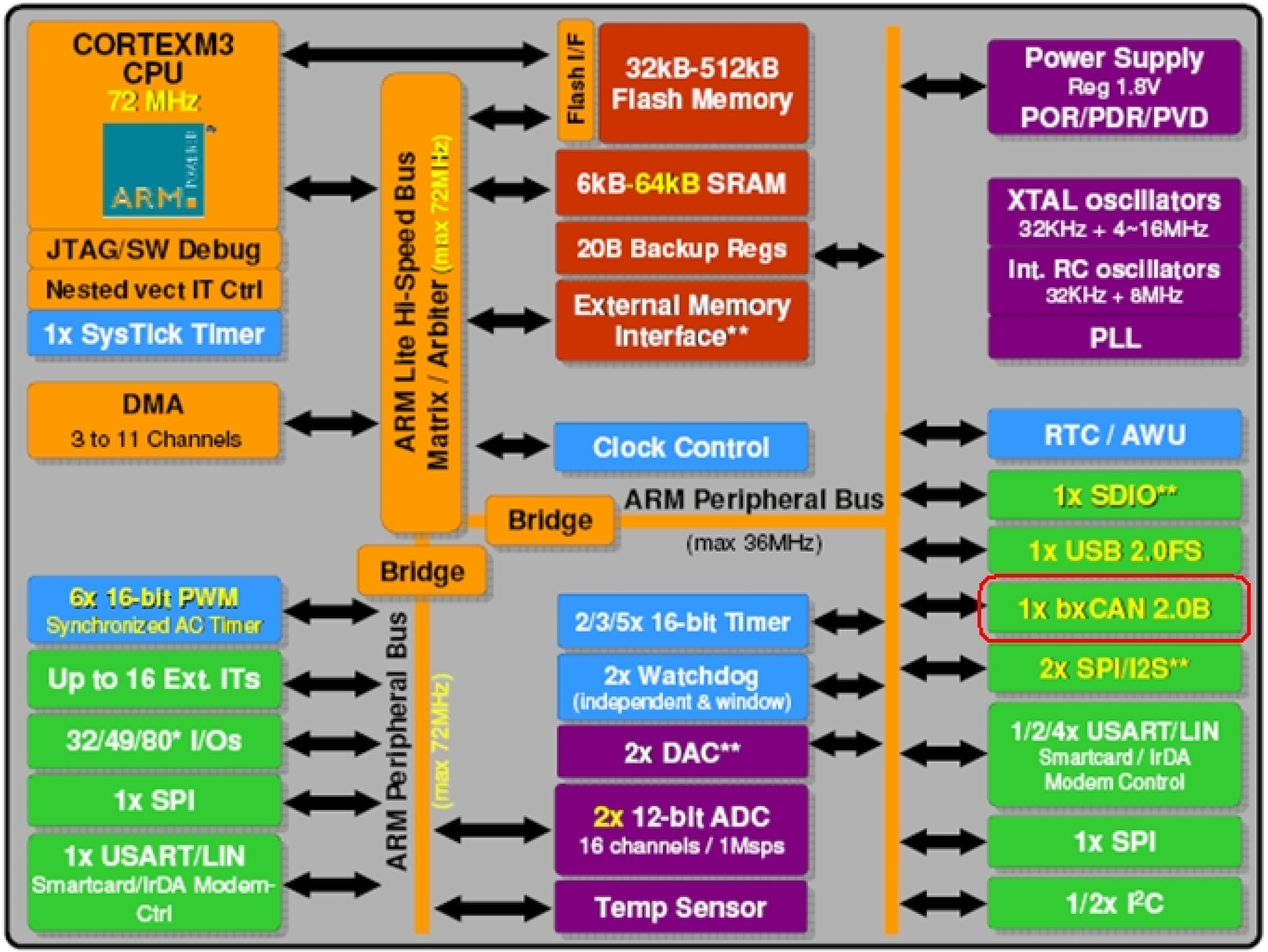
- Układ mikroprocesorowy z zewnętrznym kontrolerem CAN, transceiver CAN



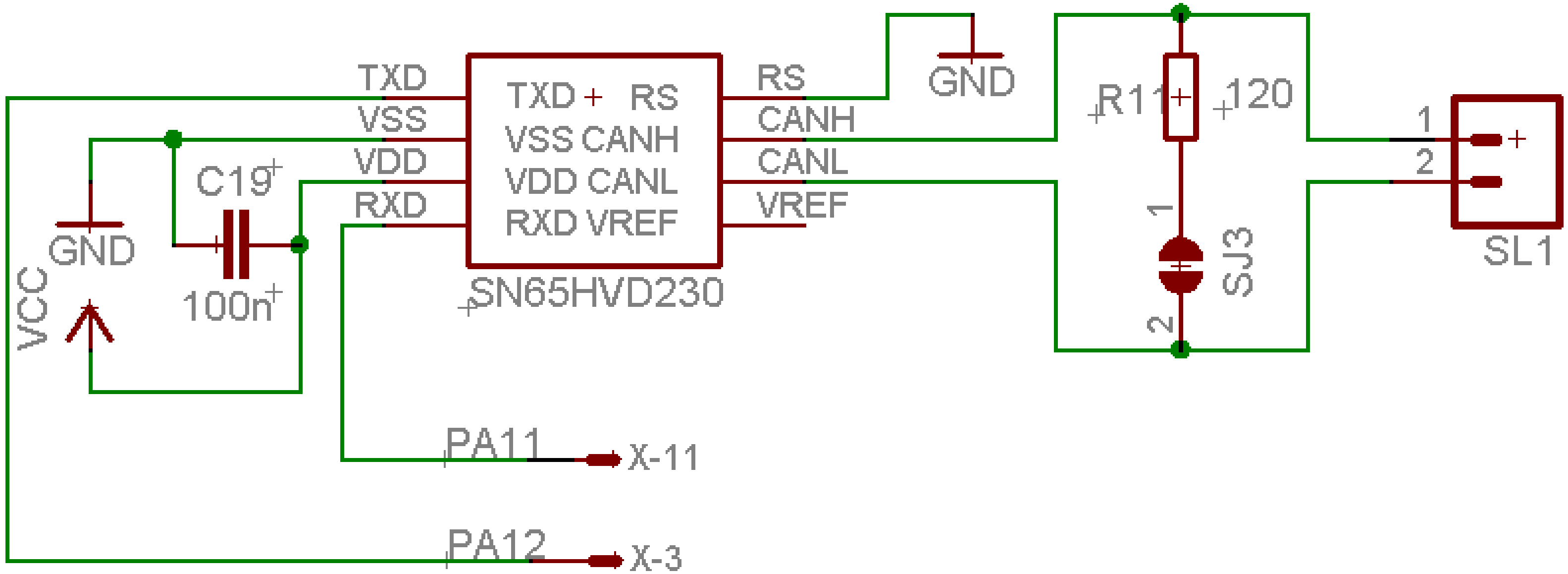
Controller CAN, transceiver CAN



CAN w STM32


CORTEXM3
CPU
72 MHz
ARM
JTAG/SW Debug
Nested vect IT Ctrl
1x SysTick Timer
DMA
3 to 11 Channels
ARM Lite Hi-Speed Bus
Matrix / Arbiter (max 72MHz)
Flash I/F
32kB-512kB
Flash Memory
6kB-64kB SRAM
20B Backup Regs
External Memory
Interface**
Clock Control
Power Supply
Reg 1.8V
POR/PDR/PVD
XTAL oscillators
32KHz + 4~16MHz
Int. RC oscillators
32KHz + 8MHz
PLL
RTC / AWU
Bridge
ARM Peripheral Bus
(max 36MHz)
1x SDIO**
1x USB 2.0FS
1x bxCAN 2.0B
Bridge
6x 16-bit PWM
Synchronized AC Timer
Up to 16 Ext. ITs
32/49/80* I/Os
1x SPI
1x USART/LIN
Smartcard/IrDA Modem-
Ctrl
ARM Peripheral Bus
(max 72MHz)
2/3/5x 16-bit Timer
2x Watchdog
(independent & window)
2x DAC**
2x 12-bit ADC
16 channels / 1Msps
Temp Sensor
2x SPI/I2S**
1/2/4x USART/LIN
Smartcard / IrDA
Modem Control
1x SPI
1/2x I²C

Schemat elektryczny



Pins					Pin name	Type	I / O Level	Main function (after reset)	Alternate functions	
BGA100	LQFP48	LQFP64	LQFP100	VFQFPN36					Default	Remap
C10	32	44	70	23	PA11	I/O	FT	PA11	USART1_CTS/ CANRX / TIM1_CH4 / USBDM	
B10	33	45	71	24	PA12	I/O	FT	PA12	USART1_RTS/ CANTX / TIM1_ETR / USBDP	

KNSW

„User manual”

UM0427
User manual

ARM®-based 32-bit MCU STM32F101xx and STM32F103xx firmware library

Introduction

This document describes the ARM®-based 32-bit MCU STM32F101xx and STM32F103xx firmware library.

This library is a firmware package which contains a collection of routines, data structures and macros covering the features of all peripherals. It includes a description of the device drivers plus a set of examples for each peripheral. The firmware library allows any device to be used in the user application without the need for in-depth study of each peripheral specifications. As a result, using the firmware library saves significant time that would otherwise be spent in coding, while reducing the application development and integration cost.

Each device driver consists of a set of functions covering all peripheral functionalities. The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions and the names of parameters.

The driver source code is developed in 'Strict ANSI-C' (relaxed ANSI-C for projects and examples files). It is fully documented and is MISRA-C 2004 compliant (the compliancy matrix is available upon request). Writing the whole library in 'Strict ANSI-C' makes it independent from the software toolchain. Only the start-up files depend on the toolchain.

The firmware library implements run-time failure detection by checking the input values for all library functions. This dynamic checking contributes to enhance the robustness of the software. Run-time detection is suitable for user application development and debugging. It adds an overhead and can be removed from the final application code to minimize code size and execution speed. For more details refer to *Section 2.5: Run-time checking on page 48*.

Since the firmware library is generic and covers all peripherals functionalities, the size and/or execution speed of the application code may not be optimized. For many applications, the library may be used as is. However, for applications having tough constraints in terms of code size and/or execution speed, the library drivers should be used as a reference on how to configure the peripheral and tailor them to specific application requirements.

The firmware library user manual is structured as follows:

- Definitions, document conventions and firmware library rules
- Overview of the firmware library (package content, library structure), installation guidelines, and example on how to use the library.
- Detailed description the firmware library: configuration structure and software functions for each peripheral.

STM32F101xx and STM32F103xx will be referred to as STM32F10xxx throughout the document.

Function name	Description
CAN_DeInit	Resets the CAN peripheral registers to their default reset values.
CAN_Init	Initializes the CAN peripheral according to the parameters specified in the CAN_InitStruct.
CAN_FilterInit	Initializes the CAN peripheral according to the parameters specified in the CAN_FilterInitStruct.
CAN_StructInit	Fills each CAN_InitStruct member with its default value.
CAN_ITConfig	Enables or disables the specified CAN interrupts.
CAN_Transmit	Initiates the transmission of a message
CAN_TransmitStatus	Checks the transmission of a message
CAN_CancelTransmit	Cancels a transmit request
CAN_FIFORelease	Releases a FIFO
CAN_MessagePending	Returns the number of pending messages
CAN_Receive	Receives a message
CAN_Sleep	Enters the low power mode
CAN_WakeUp	Wakes the CAN up
CAN_GetFlagStatus	Checks whether the specified CAN flag is set or not.
CAN_ClearFlag	Clears the CAN pending flags.
CAN_GetITStatus	Checks whether the specified CAN interrupt has occurred or not.
CAN_ClearITPendingBit	Clears the CAN interrupt pending bits.

Konfiguracja


```
typedef struct
{
  FunctionnalState CAN_TTCM;
  FunctionnalState CAN_ABOM;
  FunctionnalState CAN_AWUM;
  FunctionnalState CAN_NART;
  FunctionnalState CAN_RFLM;
  FunctionnalState CAN_TXFP;
  u8 CAN_Mode;
  u8 CAN_SJW;
  u8 CAN_BS1;
  u8 CAN_BS2;
  u16 CAN_Prescaler;
} CAN_InitTypeDef;
```

CAN_TTCM

CAN_TTCM is used to enable or disable the time triggered communication mode. This member can be set either to ENABLE or DISABLE.

CAN_ABOM

CAN_ABOM is used to enable or disable the automatic bus-off management. This member can be set either to ENABLE or DISABLE.

CAN_AWUM

CAN_AWUM is used to enable or disable the automatic wake-up mode. This member can be set either to ENABLE or DISABLE.

CAN_NART

CAN_NART is used to enable or disable the no-automatic retransmission mode. This member can be either set to ENABLE or DISABLE.

CAN_RFLM

CAN_RFLM is used to enable or disable the Receive Fifo Locked mode. This member can be either set to ENABLE or DISABLE.

CAN_TXFP

CAN_TXFP is used to enable or disable the transmit FIFO priority. This member can be set either to ENABLE or DISABLE.

CAN_Mode	Description
CAN_Mode_Normal	CAN hardware operates in normal mode
CAN_Mode_Silent	CAN hardware operates in silent mode
CAN_Mode_LoopBack	CAN hardware operates in loop back mode
CAN_Mode_Silent_LoopBack	CAN hardware operates in loop back combined with silent mode

CAN_SJW	Description
CAN_SJW_1tq	Resynchronization Jump Width=1 time quantum
CAN_SJW_2tq	Resynchronization Jump Width= 2 time quantum
CAN_SJW_3tq	Resynchronization Jump Width= 3 time quantum
CAN_SJW_4tq	Resynchronization Jump Width= 4 time quantum

CAN_BS1	Description
CAN_BS1_1tq	Bit Segment 1= 1 time quantum
...	...
CAN_BS1_16tq	Bit Segment 1= 16 time quantum

CAN_BS2	Description
CAN_BS2_1tq	Bit Segment 2= 1 time quantum
...	...
CAN_BS2_8tq	Bit Segment 2= 8 time quantum

CAN_Prescaler

CAN_Prescaler configures the length of a time quantum. It ranges from 1 to 1024.

Konfiguracja c.d.


```
typedef struct
{
  u8 CAN_FilterNumber;
  u8 CAN_FilterMode;
  u8 CAN_FilterScale;
  u16 CAN_FilterIdHigh;
  u16 CAN_FilterIdLow;
  u16 CAN_FilterMaskIdHigh;
  u16 CAN_FilterMaskIdLow;
  u16 CAN_FilterFIFOAssignment;
  FunctionalState CAN_FilterActivation;
} CAN_FilterInitTypeDef;
```

CAN_FilterIdHigh

CAN_FilterIdHigh is used to select the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterIdLow

CAN_FilterIdLow is used to select the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterFIFO	Description
CAN_FilterFIFO0	Filter FIFO 0 assignment for filter x
CAN_FilterFIFO1	Filter FIFO 1assignment for filter x

CAN_FilterNumber

CAN_FilterNumber selects the filter which will be initialized. It ranges from 0 to 13.

CAN_FilterMode	Description
CAN_FilterMode_IdMask	id/mask mode
CAN_FilterMode_IdList	identifier list mode

CAN_FilterScale	Description
CAN_FilterScale_Two16bit	Two 16-bit filters
CAN_FilterScale_One32bit	One 32-bit filter

CAN_FilterMaskIdHigh

CAN_FilterMaskIdHigh is used to select the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterMaskIdLow

CAN_FilterMaskIdLow is used to select the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterActivation

CAN_FilterActivation enables or disables the filter. It can be set either to ENABLE or DISABLE.

Przykład


```

CAN_InitTypeDef         CAN_InitStructure;
CAN_FilterInitTypeDef   CAN_FilterInitStructure;
CanTxMsg TxMessage;
CanRxMsg RxMessage;
u8 TransmitMailbox;

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN, ENABLE);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
CAN_InitStructure.CAN_TTCM=ENABLE;
CAN_InitStructure.CAN_ABOM=ENABLE;
CAN_InitStructure.CAN_AWUM=ENABLE;
CAN_InitStructure.CAN_NART=ENABLE;
CAN_InitStructure.CAN_RFLM=ENABLE;
CAN_InitStructure.CAN_TXFP=ENABLE;
CAN_InitStructure.CAN_Mode=CAN_Mode_Normal;
CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
CAN_InitStructure.CAN_BS1=CAN_BS1_8tq;
CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;
CAN_InitStructure.CAN_Prescaler=5;
CAN_Init(&CAN_InitStructure);

CAN_FilterInitStructure.CAN_FilterNumber=0;
CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
CAN_FilterInit(&CAN_FilterInitStructure);
```

KNSW

Przykład c.d.


```
TxMessage.StdId=0x24;
TxMessage.RTR=CAN_RTR_DATA;
TxMessage.IDE=CAN_ID_STD;
TxMessage.DLC=2;
TxMessage.Data[0]=0xCA;
TxMessage.Data[1]=0xFE;

TransmitMailbox=CAN_Transmit(&TxMessage);
while((CAN_TransmitStatus(TransmitMailbox) != CANTXOK) && (i != 0xFF))
{
}
```

```
RxMessage.StdId=0x00;
RxMessage.IDE=CAN_ID_STD;
RxMessage.DLC=0;
RxMessage.Data[0]=0x00;
RxMessage.Data[1]=0x00;

while (CAN_MessagePending(CAN_FIFO0))
{
  CAN_Receive(CAN_FIFO0, &RxMessage);
}

while (CAN_MessagePending(CAN_FIFO1))
{
  CAN_Receive(CAN_FIFO0, &RxMessage2);
}
```

Koniec