# CAN-BUS Data Frames

To watch CAN frames can be frustrative if you connect an oscilloscope with the red pin (14) and ground (GND). You get probably this - *Image 2* :

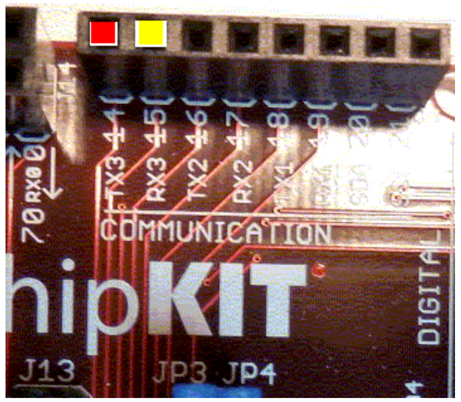


*Image 1 with CAN1*

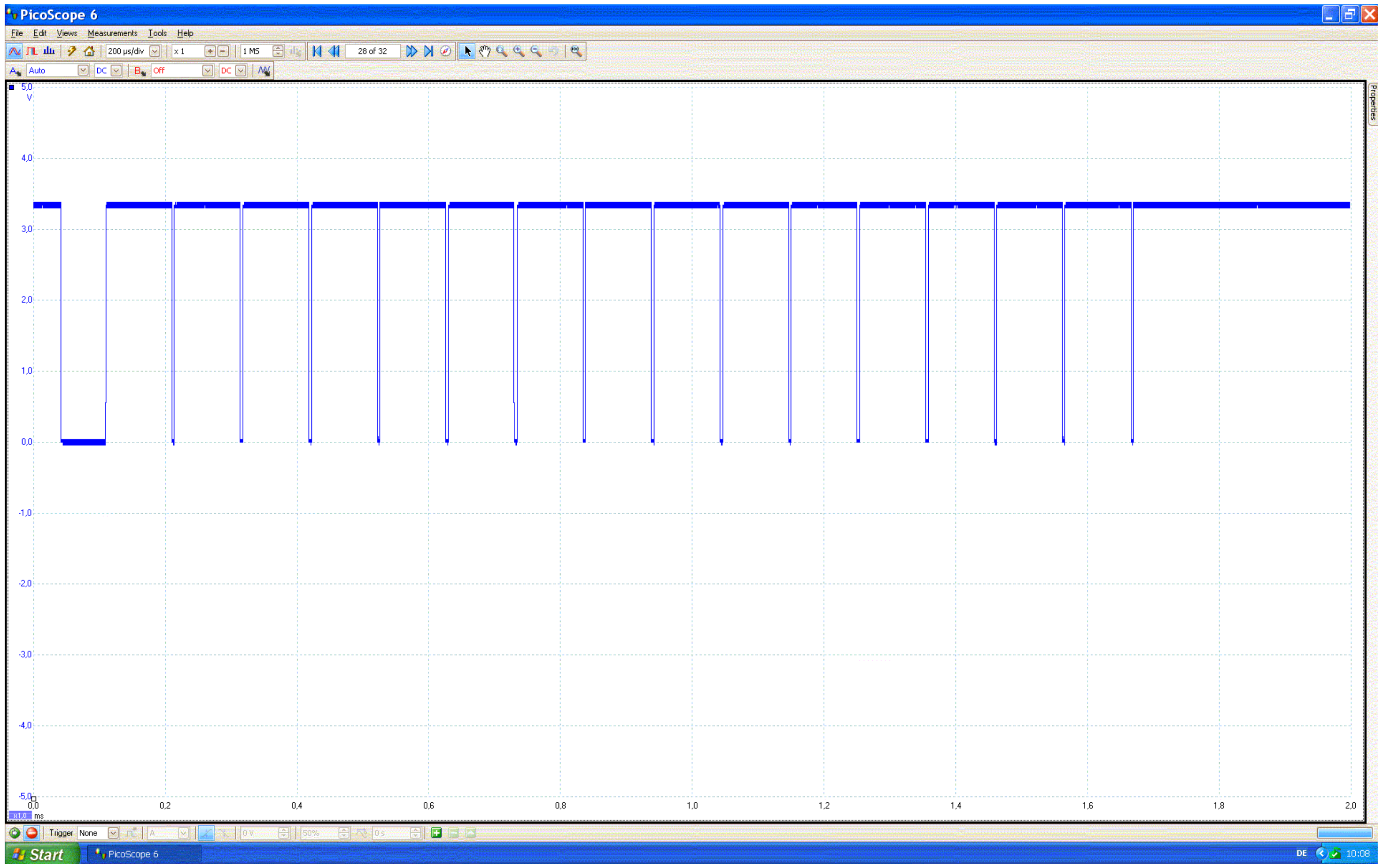


*Image 2*

… but that's not what you expected; you expected probably this - *Image 3* :

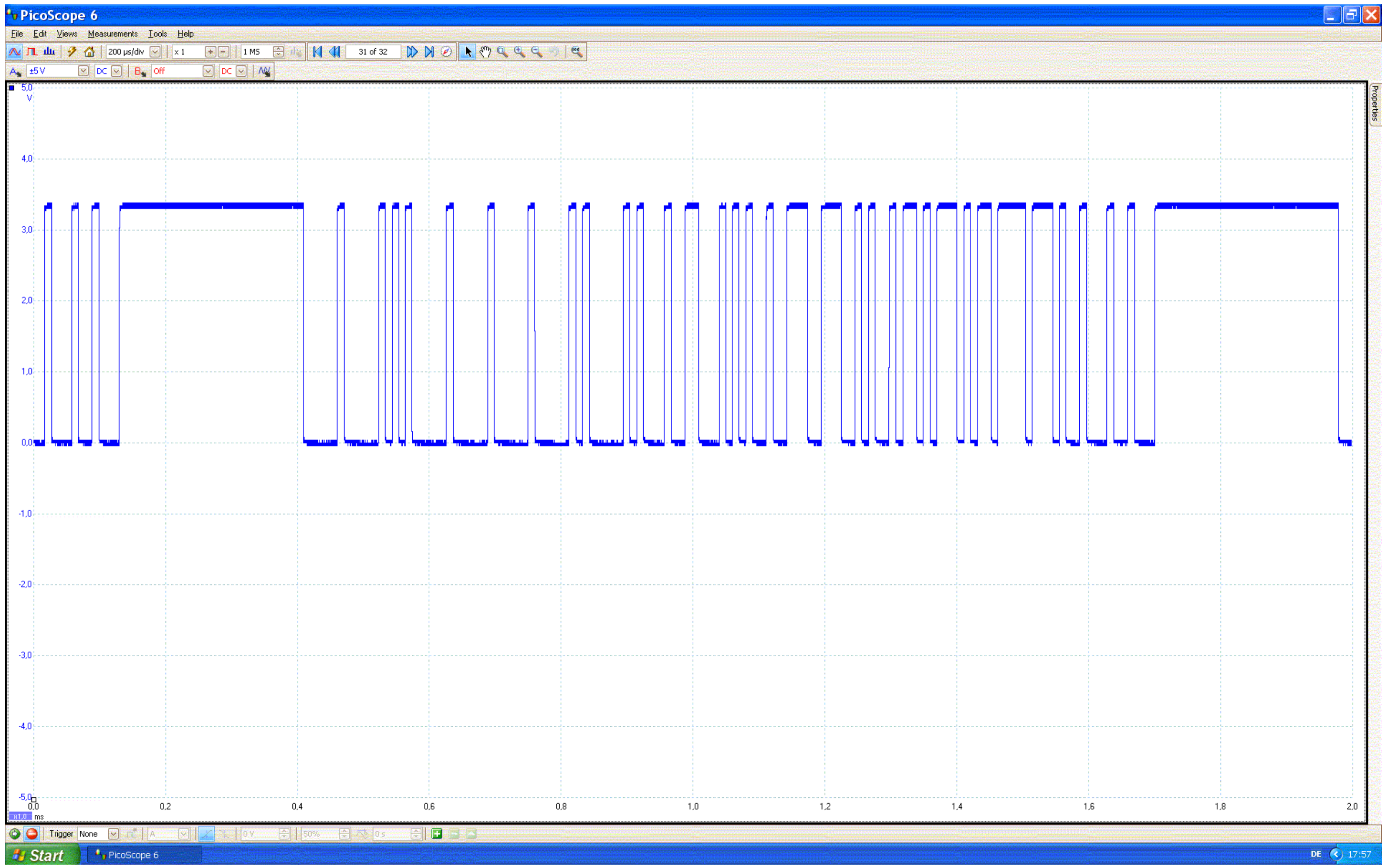


*Image 3*

To get the result of *Image 3*, you have to connect the red and yellow pins (14 and 15) of *Image 1* with a resistor $0\Omega < R < 100k\Omega$. The value can be in a wide range.

Now you can connect the oscilloscope with the red pin (14) and you will get what you want - *Image 4* .

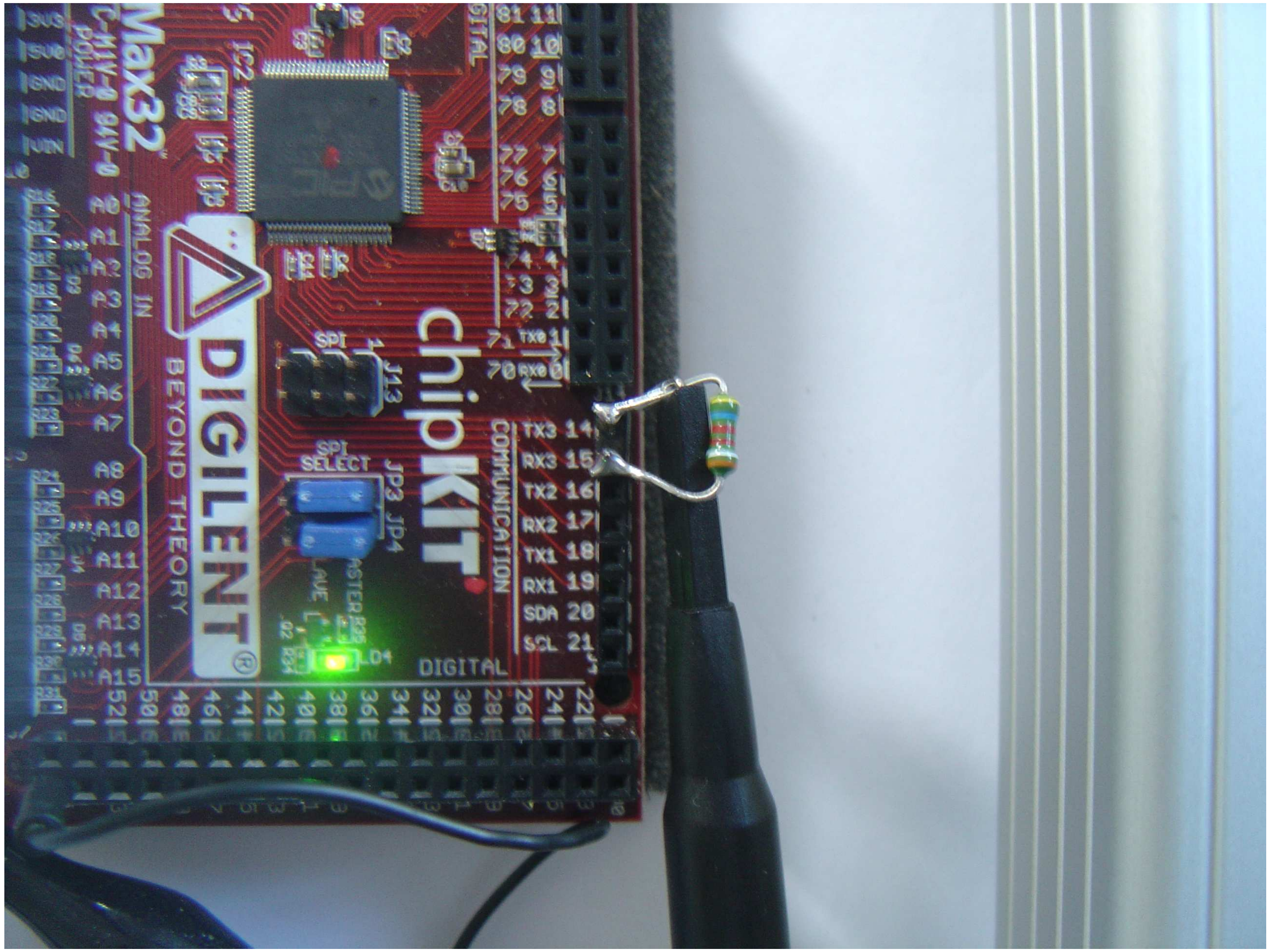


*Image 4 - CAN1 with a 39kΩ resistor*

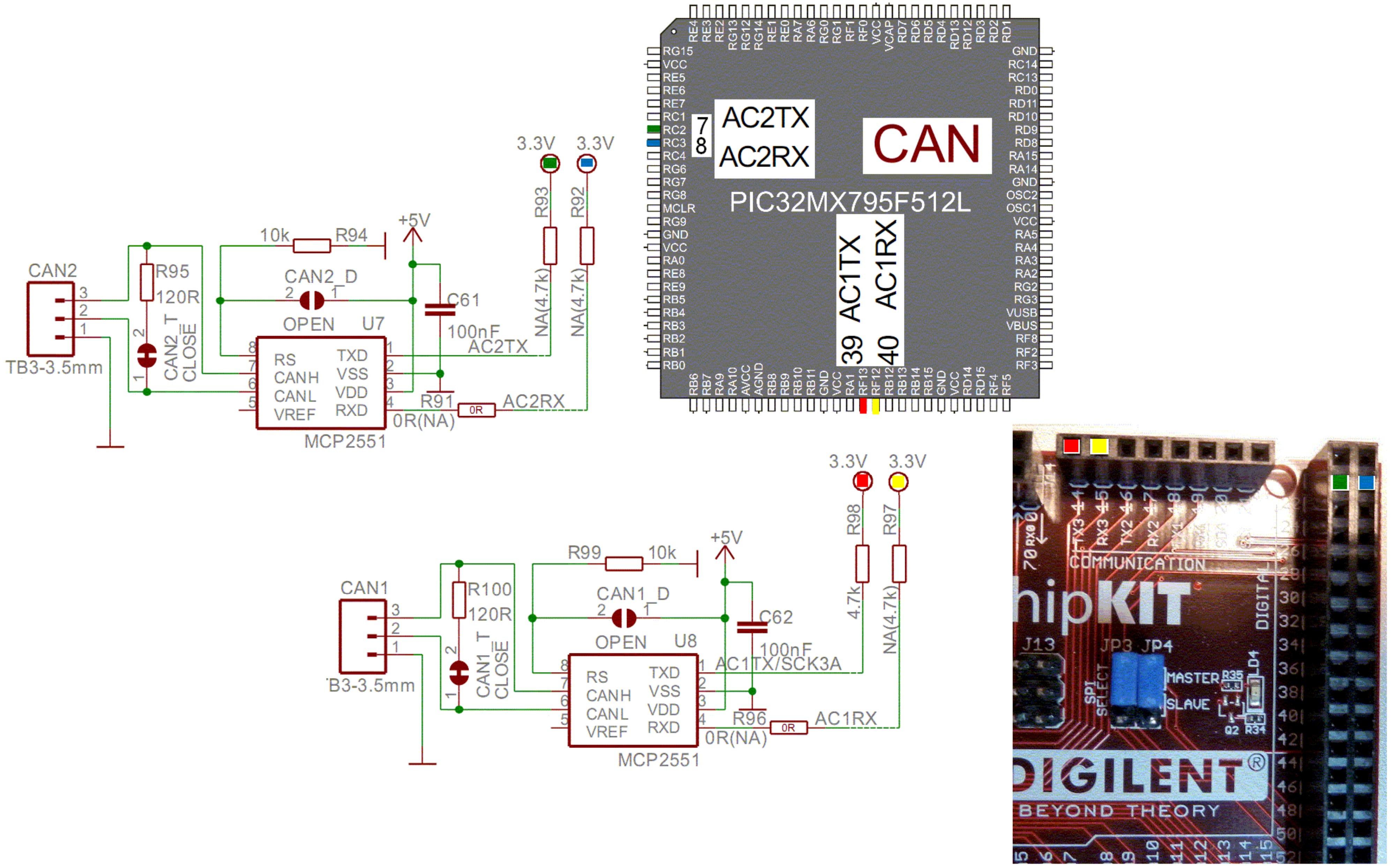


*Image 5 :* *Who Is Where ?* *You can work with R96=R97=R98=R99=0Ω (Zero Ohm)* *CAN1* *CAN2*

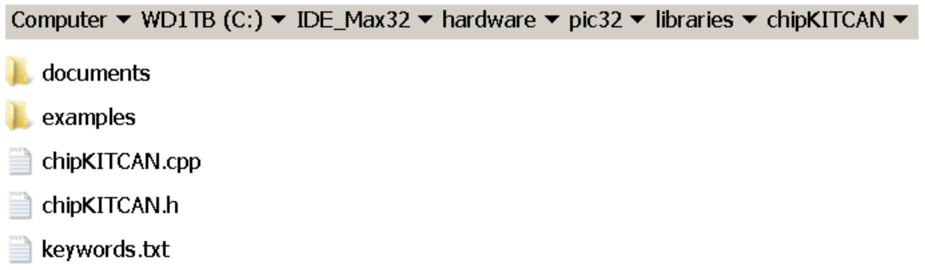
A test program for *Image 3* :

```
#include  <WProgram.h>
#include  "chipKITCAN.h"

#define SYS_FREQ(80000000L)
#define CAN_BUS_SPEED   100000              // CAN Speed
CAN    canMod1(CAN::CAN1);    // this object uses CAN module 1
uint8_t  CAN1MessageFifoArea[2 * 8 * 16];
void initCan1();
void txCAN1();

void setup() {   initCan1();   }
void loop() {   txCAN1();    }

void initCan1() {
  CAN::BIT_CONFIG canBitConfig;
  canMod1.enableModule(TRUE);
  canMod1.setOperatingMode(CAN::CONFIGURATION);
  while(canMod1.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq          = CAN::BIT_2TQ;
  canBitConfig.phaseSeg1Tq          = CAN::BIT_2TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_2TQ;
```

Download the software for the chipKIT Network Shield and put

chipKITCAN into: C:\IDE_Max32\hardware\pic32\libraries.

Then it looks like:



Computer ▾ WD1TB (C:) ▾ IDE_Max32 ▾ hardware ▾ pic32 ▾ libraries ▾ chipKITCAN ▾

documents
examples
chipKITCAN.cpp
chipKITCAN.h
keywords.txt

```
  canBitConfig.phaseSeg2TimeSelect    = FALSE;
  canBitConfig.sample3Time            = FALSE;
  canBitConfig.syncJumpWidth          = CAN::BIT_1TQ;  //1 to 4 Time_Quanta
  canMod1.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod1.assignMemoryBuffer(CAN1MessageFifoArea,2 * 8 * 16);
  canMod1.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::HIGHEST_PRIORITY);
  canMod1.setOperatingMode(CAN::NORMAL_OPERATION);
  while(canMod1.getOperatingMode() != CAN::NORMAL_OPERATION);
}

void txCAN1() {
  CAN::TxMessageBuffer * message;
  message = canMod1.getTxMessageBuffer(CAN::CHANNEL0);
  if (message != NULL) {
    message->msgEID.EID   = 1234;      //receiving node
    message->msgEID.IDE   = 1;
    message->msgEID.SRR   = 1;
    message->msgEID.RTR   = 0;
    message->msgEID.DLC   = 8;
    message->data[0]      = 0x11;
    message->data[1]      = 0x31;
    message->data[2]      = 0x52;
    message->data[3]      = 0x73;
    message->data[4]     = 0x94;
    message->data[5]     = 0xB5;
    message->data[6]     = 0xD6;
    message->data[7]     = 0xF7;
    canMod1.updateChannel(CAN::CHANNEL0);
    canMod1.flushTxChannel(CAN::CHANNEL0);
  }
}
```

## Loopback Mode

Loopback mode is used for self-test to allow the CAN module to receive its own message. In this mode, the CAN module transmit path is connected internally to the receive path. A “dummy” Acknowledge is provided thereby eliminating the need for another node to provide the Acknowledge bit. The CAN message is not actually transmitted on the CAN bus, and you can see nothing but a straight line at the red pin(14) or green pin(22) with the oscilloscope - see *image 5* for the location of the green pin(22). The following program demonstrates a self-test.

```
#include  <WProgram.h>
#include  "chipKITCAN.h"

#define CAN_1_ID  0x123 //CAN module transmit path is connected internally to the receive path
#define CAN_2_ID  0x123 //--> CAN_1_ID == CAN_2_ID in this program
```

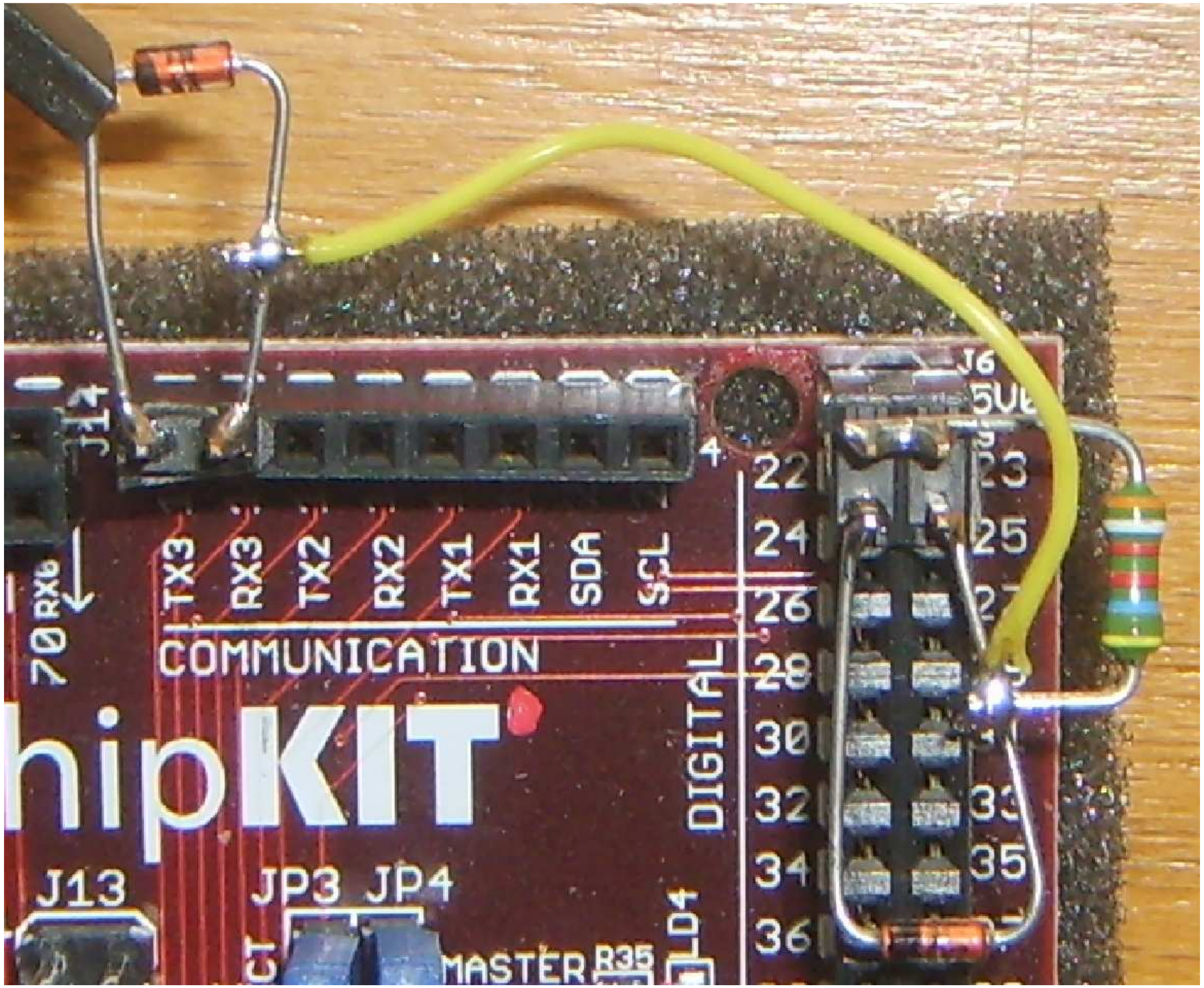
```
#define SYS_FREQ(80000000L)
#define CAN_BUS_SPEED   250000              // CAN Speed

CAN    canMod1(CAN::CAN1);    // this object uses CAN module 1
CAN    canMod2(CAN::CAN2);    // this object uses CAN module 2

uint8_t  CAN1MessageFifoArea[2 * 8 * 16];
uint8_t  CAN2MessageFifoArea[2 * 8 * 16];

static volatile bool isCAN1MsgReceived = false;
static volatile bool isCAN2MsgReceived = false;

void initCan1(uint32_t myaddr);
void initCan2(uint32_t myaddr);
void txCAN1(uint32_t rxnode);
void txCAN2(uint32_t rxnode);
void rxCAN1(void);
void rxCAN2(void);
void doCan1Interrupt();
void doCan2Interrupt();

void setup() {
  initCan1(CAN_1_ID);
  initCan2(CAN_2_ID);
  canMod1.attachInterrupt(doCan1Interrupt);
  canMod2.attachInterrupt(doCan2Interrupt);
  Serial.begin(9600);
}

void loop() {
  txCAN2(CAN_1_ID);
  delay(100);    //wait so that the character has time to be delivered
  rxCAN1();
  txCAN1(CAN_2_ID);
  delay(100);
  rxCAN2();
}

void initCan1(uint32_t myaddr) {
  CAN::BIT_CONFIG canBitConfig;
  canMod1.enableModule(TRUE);
  canMod1.setOperatingMode(CAN::CONFIGURATION);
  while(canMod1.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq           = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq           = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect    = TRUE;
  canBitConfig.sample3Time            = TRUE;
  canBitConfig.syncJumpWidth          = CAN::BIT_2TQ;
  canMod1.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod1.assignMemoryBuffer(CAN1MessageFifoArea,2 * 8 * 16);
  canMod1.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::LOW_MEDIUM_PRIORITY);
  canMod1.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE);
  canMod1.configureFilter      (CAN::FILTER0, myaddr, CAN::SID);
  canMod1.configureFilterMask  (CAN::FILTER_MASK0, 0xFFF, CAN::SID, CAN::FILTER_MASK_IDE_TYPE);
  canMod1.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod1.enableFilter         (CAN::FILTER0, TRUE);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
  canMod1.enableModuleEvent(CAN::RX_EVENT, TRUE);
  canMod1.setOperatingMode(CAN::LOOPBACK);
  while(canMod1.getOperatingMode() != CAN::LOOPBACK);
}

void initCan2(uint32_t myaddr) {
  CAN::BIT_CONFIG canBitConfig;
  canMod2.enableModule(TRUE);
  canMod2.setOperatingMode(CAN::CONFIGURATION);
  while(canMod2.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq            = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq           = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq      = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect   = TRUE;
  canBitConfig.sample3Time           = TRUE;
  canBitConfig.syncJumpWidth         = CAN::BIT_2TQ;
  canMod2.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod2.assignMemoryBuffer(CAN2MessageFifoArea,2 * 8 * 16);
  canMod2.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::LOW_MEDIUM_PRIORITY);
  canMod2.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE);
  canMod2.configureFilter     (CAN::FILTER0, myaddr, CAN::SID);
  canMod2.configureFilterMask  (CAN::FILTER_MASK0, 0xFFF, CAN::SID, CAN::FILTER_MASK_IDE_TYPE);
  canMod2.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod2.enableFilter        (CAN::FILTER0, TRUE);
  canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
  canMod2.enableModuleEvent(CAN::RX_EVENT, TRUE);
  canMod2.setOperatingMode(CAN::LOOPBACK);
  while(canMod2.getOperatingMode() != CAN::LOOPBACK);
}

void txCAN1(uint32_t rxnode) {
  CAN::TxMessageBuffer * message;
  message = canMod1.getTxMessageBuffer(CAN::CHANNEL0);
  if (message != NULL) {
    message->messageWord[0] = 0;
    message->messageWord[1] = 0;
    message->messageWord[2] = 0;
    message->messageWord[3] = 0;
    message->msgSID.SID   = rxnode;    //receiving node
    message->msgEID.IDE   = 0;
    message->msgEID.RTR   = 0;
    message->msgEID.SRR   = 0;
    message->msgEID.DLC   = 1;
    message->data[0]      = 12;
    //message->data[1]        = 0x31;
    //message->data[2]        = 0x32;
    //message->data[3]        = 0x33;
    //message->data[4]        = 0x34;
    //message->data[5]        = 0x35;
    //message->data[6]        = 0x36;
    //message->data[7]        = 0x37;
    canMod1.updateChannel(CAN::CHANNEL0);
    canMod1.flushTxChannel(CAN::CHANNEL0);
  }
}

void txCAN2(uint32_t rxnode) {
```

```
CAN::TxMessageBuffer * message;
message = canMod2.getTxMessageBuffer(CAN::CHANNEL0);
if (message != NULL) {
  message->messageWord[0] = 0;
  message->messageWord[1] = 0;
  message->messageWord[2] = 0;
  message->messageWord[3] = 0;
  message->msgSID.SID    = rxnode;   //receiving node
  message->msgEID.IDE    = 0;
  message->msgEID.RTR    = 0;
  message->msgEID.SRR    = 0;
  message->msgEID.DLC    = 1;
  message->data[0]       = 34;
  //message->data[1]       = 0x31;
  //message->data[2]       = 0x32;
  //message->data[3]       = 0x33;
  //message->data[4]       = 0x34;
  //message->data[5]       = 0x35;
  //message->data[6]       = 0x36;
  //message->data[7]       = 0x37;
  canMod2.updateChannel(CAN::CHANNEL0);
  canMod2.flushTxChannel(CAN::CHANNEL0);
}
```

```
}

void rxCAN1(void) {
  CAN::RxMessageBuffer * message;
  if (isCAN1MsgReceived == false) {
    return;
  }
  isCAN1MsgReceived = false;
  message = canMod1.getRxMessage(CAN::CHANNEL1);
  Serial.println("CAN1: The CAN_1_2 message is not actually transmitted on the CAN bus");
  canMod1.updateChannel(CAN::CHANNEL1);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
}

void rxCAN2(void) {
  CAN::RxMessageBuffer * message;
  if (isCAN2MsgReceived == false) {
      return;
  }
  isCAN2MsgReceived = false;
  message = canMod2.getRxMessage(CAN::CHANNEL1);
  Serial.println("CAN2: no CAN-Frame for an oscilloscope at pin(14) and pin(22)");
  canMod2.updateChannel(CAN::CHANNEL1);
  canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
}

void doCan1Interrupt() {
  if ((canMod1.getModuleEvent() & CAN::RX_EVENT) != 0) {
    if(canMod1.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
      canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, FALSE);
      isCAN1MsgReceived = true;
    }
  }
}

void doCan2Interrupt() {
  if ((canMod2.getModuleEvent() & CAN::RX_EVENT) != 0) {
    if(canMod2.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
           canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, FALSE);
      isCAN2MsgReceived = true;
    }
  }
}
```

## Let's communicate CAN1 with CAN2 ...

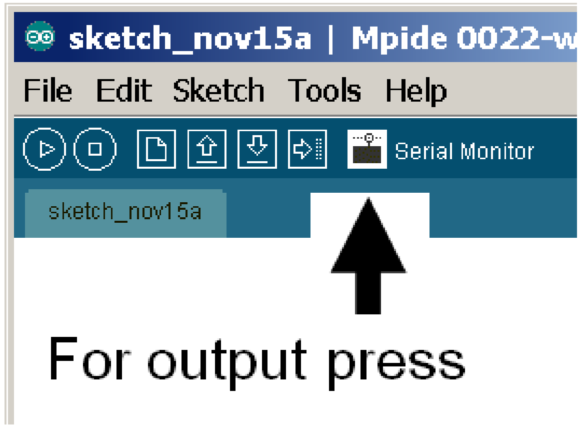
We will need a little setup for that, composed of two diods and a restistor. I use 39.2kΩ, but 5kΩ to 50kΩ are working as well.



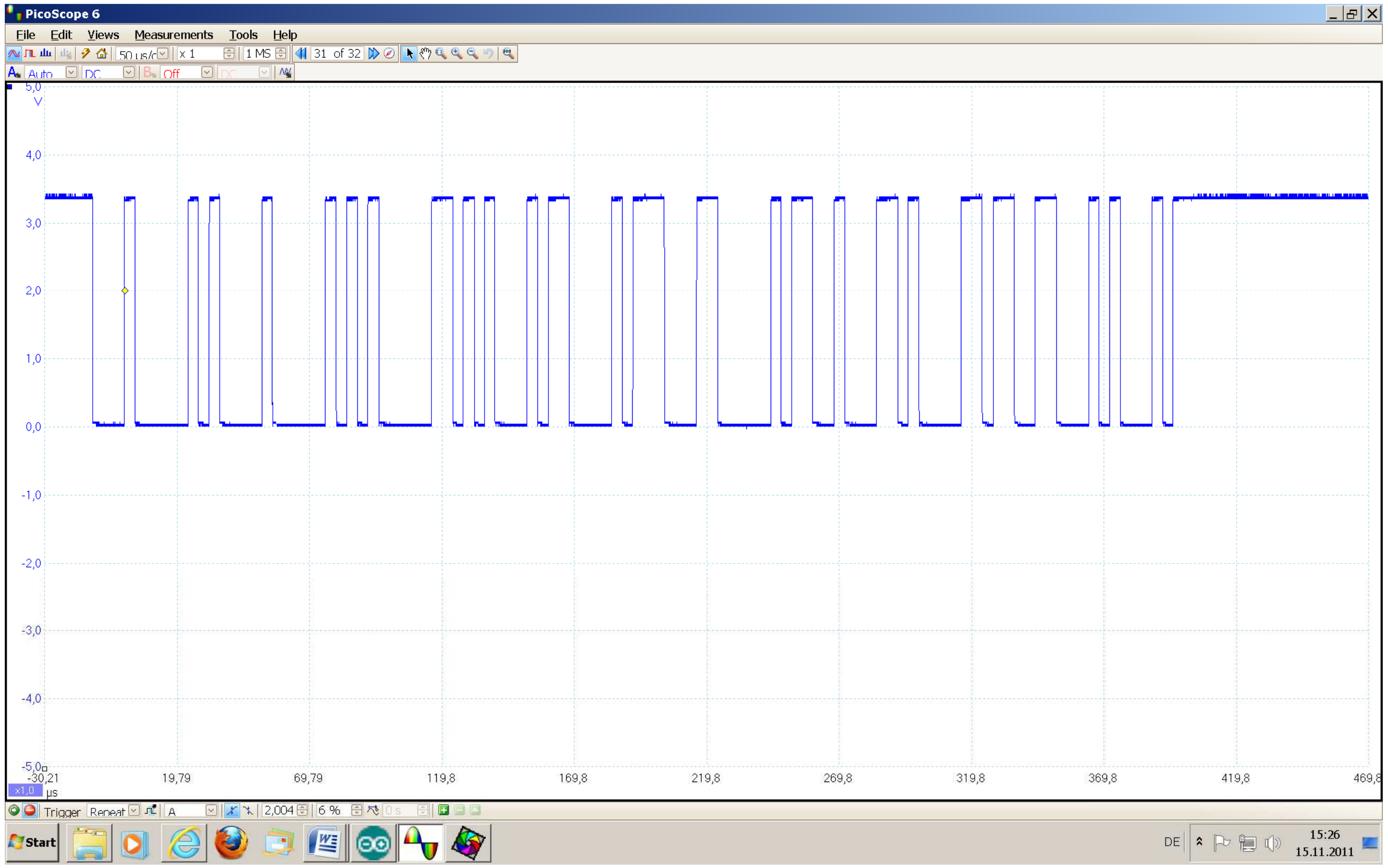
```
#include  <WProgram.h>
#include  "chipKITCAN.h"

#define node1can1     0x101L
#define node1can2     0x102L

#define SYS_FREQ (80000000L)
#define CAN_BUS_SPEED   250000         // CAN Speed

int data[8];

CAN    canMod1(CAN::CAN1);    // this object uses CAN module 1
CAN    canMod2(CAN::CAN2);    // this object uses CAN module 2

uint8_t  CAN1MessageFifoArea[2 * 8 * 16];
uint8_t  CAN2MessageFifoArea[2 * 8 * 16];

static volatile bool isCAN1MsgReceived = false;
static volatile bool isCAN2MsgReceived = false;

void initCan1(uint32_t myaddr);
void initCan2(uint32_t myaddr);
void txCAN1(uint32_t rxnode);
void txCAN2(uint32_t rxnode);
void rxCAN1(void);
void rxCAN2(void);
void doCan1Interrupt();
void doCan2Interrupt();

void setup() {
  initCan1(node1can1);
  initCan2(node1can2);
  canMod1.attachInterrupt(doCan1Interrupt);
  canMod2.attachInterrupt(doCan2Interrupt);
  Serial.begin(9600);
}

void loop() {
  txCAN2(node1can1);
  delay(100);    //wait so that the character has time to be delivered
  rxCAN1();
  txCAN1(node1can2);
  delay(100);
  rxCAN2();
}
```

```
void initCan1(uint32_t myaddr) {
  CAN::BIT_CONFIG canBitConfig;
  canMod1.enableModule(TRUE);
  canMod1.setOperatingMode(CAN::CONFIGURATION);
  while(canMod1.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq            = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq            = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect    = TRUE;
  canBitConfig.sample3Time            = TRUE;
  canBitConfig.syncJumpWidth          = CAN::BIT_2TQ;
  canMod1.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod1.assignMemoryBuffer(CAN1MessageFifoArea,2 * 8 * 16);
  canMod1.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::LOW_MEDIUM_PRIORITY);
  canMod1.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE); //RX_DATA_ONLY
  canMod1.configureFilter      (CAN::FILTER0, myaddr, CAN::SID);
  canMod1.configureFilterMask  (CAN::FILTER_MASK0, 0xFFF, CAN::SID, CAN::FILTER_MASK_IDE_TYPE);
  canMod1.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod1.enableFilter         (CAN::FILTER0, TRUE);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
  canMod1.enableModuleEvent(CAN::RX_EVENT, TRUE);
  canMod1.setOperatingMode(CAN::NORMAL_OPERATION);
  while(canMod1.getOperatingMode() != CAN::NORMAL_OPERATION);
}

void initCan2(uint32_t myaddr) {
  CAN::BIT_CONFIG canBitConfig;
  canMod2.enableModule(TRUE);
  canMod2.setOperatingMode(CAN::CONFIGURATION);
  while(canMod2.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq            = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq            = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect    = TRUE;
  canBitConfig.sample3Time            = TRUE;
  canBitConfig.syncJumpWidth          = CAN::BIT_2TQ;
  canMod2.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod2.assignMemoryBuffer(CAN2MessageFifoArea,2 * 8 * 16);
  canMod2.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::LOW_MEDIUM_PRIORITY);
  canMod2.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE);
  canMod2.configureFilter      (CAN::FILTER0, myaddr, CAN::SID);
  canMod2.configureFilterMask  (CAN::FILTER_MASK0, 0xFFF, CAN::SID, CAN::FILTER_MASK_IDE_TYPE);
  canMod2.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod2.enableFilter         (CAN::FILTER0, TRUE);
  canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
  canMod2.enableModuleEvent(CAN::RX_EVENT, TRUE);
  canMod2.setOperatingMode(CAN::NORMAL_OPERATION);
  while(canMod2.getOperatingMode() != CAN::NORMAL_OPERATION);
```

```
}

void txCAN1(uint32_t rxnode) {
  CAN::TxMessageBuffer *message;
  message = canMod1.getTxMessageBuffer(CAN::CHANNEL0);
  if (message != NULL) {
    message->messageWord[0] = 0;
    message->messageWord[1] = 0;
    message->messageWord[2] = 0;
    message->messageWord[3] = 0;
    message->msgSID.SID     = rxnode;  //receiving node
    message->msgEID.RTR     = 0;
    message->msgEID.SRR     = 1;
    message->msgEID.IDE     = 0;
    message->msgEID.DLC     = 8;
    message->data[0]        = 20;
    message->data[1]        = 21;
    message->data[2]        = 22;
    message->data[3]        = 23;
    message->data[4]        = 24;
    message->data[5]        = 25;
    message->data[6]        = 26;
    message->data[7]        = 27;
    canMod1.updateChannel(CAN::CHANNEL0);
    canMod1.flushTxChannel(CAN::CHANNEL0);
  }
}

void txCAN2(uint32_t rxnode) {
  CAN::TxMessageBuffer *message;
  message = canMod2.getTxMessageBuffer(CAN::CHANNEL0);
  if (message != NULL) {
    message->messageWord[0] = 0;
    message->messageWord[1] = 0;
    message->messageWord[2] = 0;
    message->messageWord[3] = 0;
    message->msgSID.SID     = rxnode;  //receiving node
    message->msgEID.RTR     = 0;
    message->msgEID.SRR     = 1;
    message->msgEID.IDE     = 0;
    message->msgEID.DLC     = 8;
    message->data[0]        = 40;
    message->data[1]        = 41;
    message->data[2]        = 42;
    message->data[3]        = 43;
    message->data[4]        = 44;
    message->data[5]        = 45;
    message->data[6]        = 46;
    message->data[7]        = 47;
    canMod2.updateChannel(CAN::CHANNEL0);
    canMod2.flushTxChannel(CAN::CHANNEL0);
  }
}

void rxCAN1(void) {
  int k;
  CAN::RxMessageBuffer *message;
  if (isCAN1MsgReceived == false) {
    return;
  }
  isCAN1MsgReceived = false;
  message = canMod1.getRxMessage(CAN::CHANNEL1);
  for(k=0;k<8;k++)
    data[k] = message->data[k];
  for(k=0;k<7;k++){
    Serial.print(data[k]);
    Serial.print(", ");
  }
  Serial.println(data[7]);
  canMod1.updateChannel(CAN::CHANNEL1);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
}

void rxCAN2(void) {
  int k;
  CAN::RxMessageBuffer *message;
  if (isCAN2MsgReceived == false) {
      return;
  }
  isCAN2MsgReceived = false;
  message = canMod2.getRxMessage(CAN::CHANNEL1);
  for(k=0;k<8;k++)
    data[k] = message->data[k];
  for(k=0;k<7;k++){
    Serial.print(data[k]);
    Serial.print(", ");
  }
  Serial.println(data[7]);
  canMod2.updateChannel(CAN::CHANNEL1);
  canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
}

void doCan1Interrupt() {
  if ((canMod1.getModuleEvent() & CAN::RX_EVENT) != 0) {
    if(canMod1.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
      canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, FALSE);
```

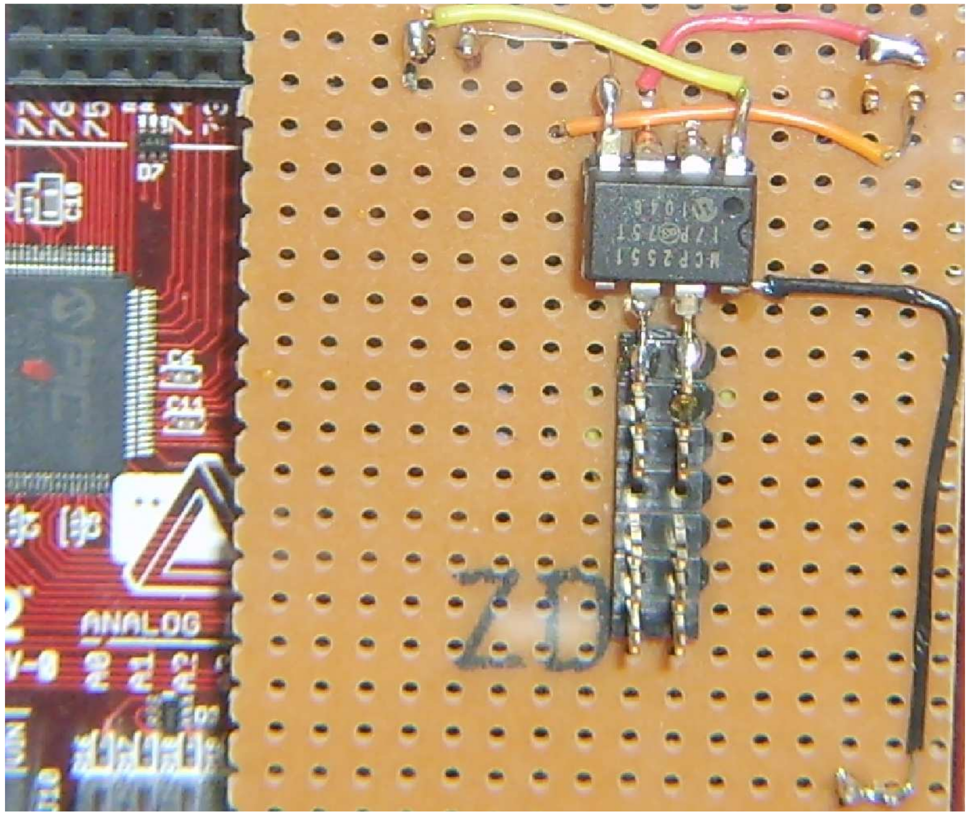


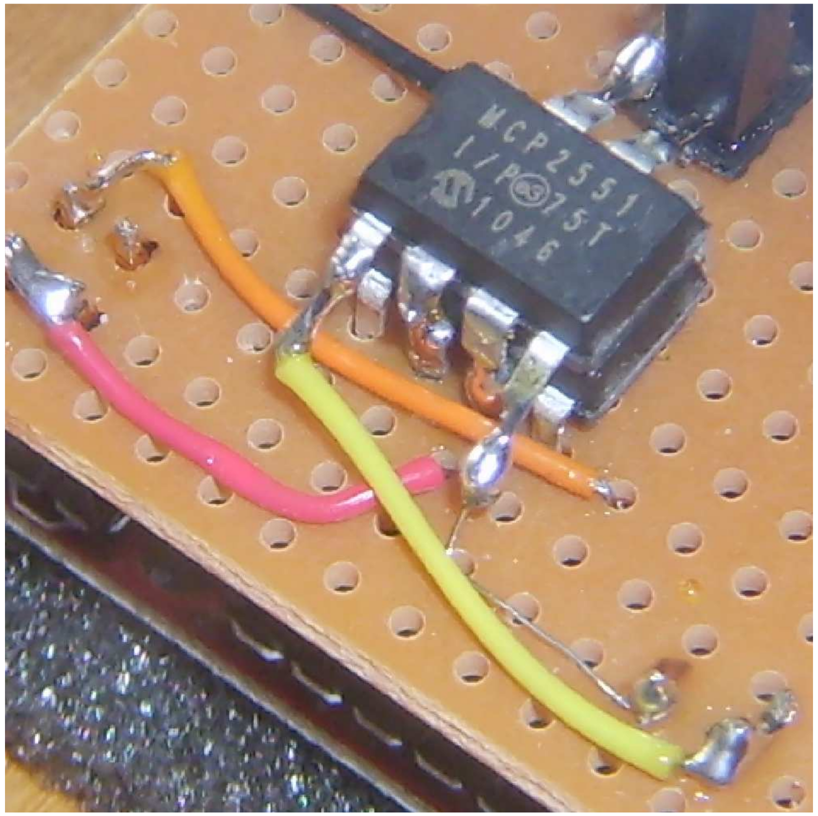
```
      isCAN1MsgReceived = true;
    }
  }
}

void doCan2Interrupt() {
  if ((canMod2.getModuleEvent() & CAN::RX_EVENT) != 0) {
    if(canMod2.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
      canMod2.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, FALSE);
      isCAN2MsgReceived = true;
    }
  }
}

/*
40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

40, 41, 42, 43, 44, 45, 46, 47

20, 21, 22, 23, 24, 25, 26, 27

*/
```



## CAN_communicate with the outside world ...

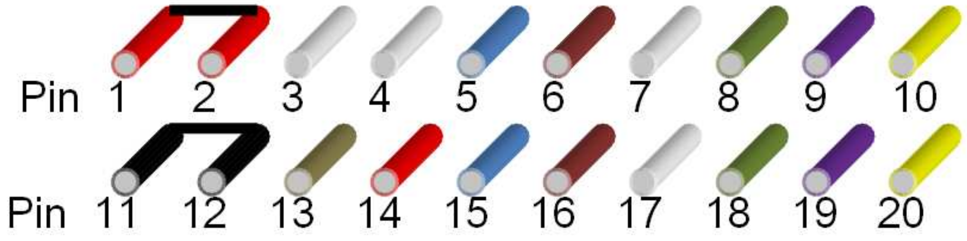
To communicate with the outside world of the MAX_32 board we need a CAN_transceiver, as shown in ***Image 5.***

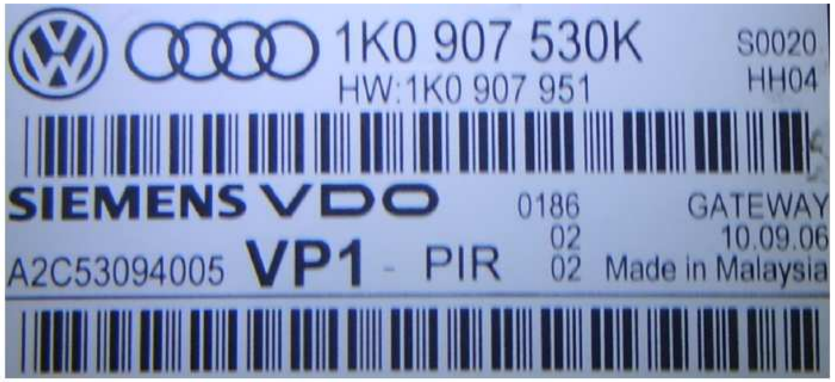




Our partner is VAG/AUDI/VW Gateway: 1K0 907 530K.

The gateway is calling for other devices: instrument cluster, radio, navi ..., but nothing of them is connected to the gateway on my desktop.





1K0 907 530K S0020
HW:1K0 907 951 HH04

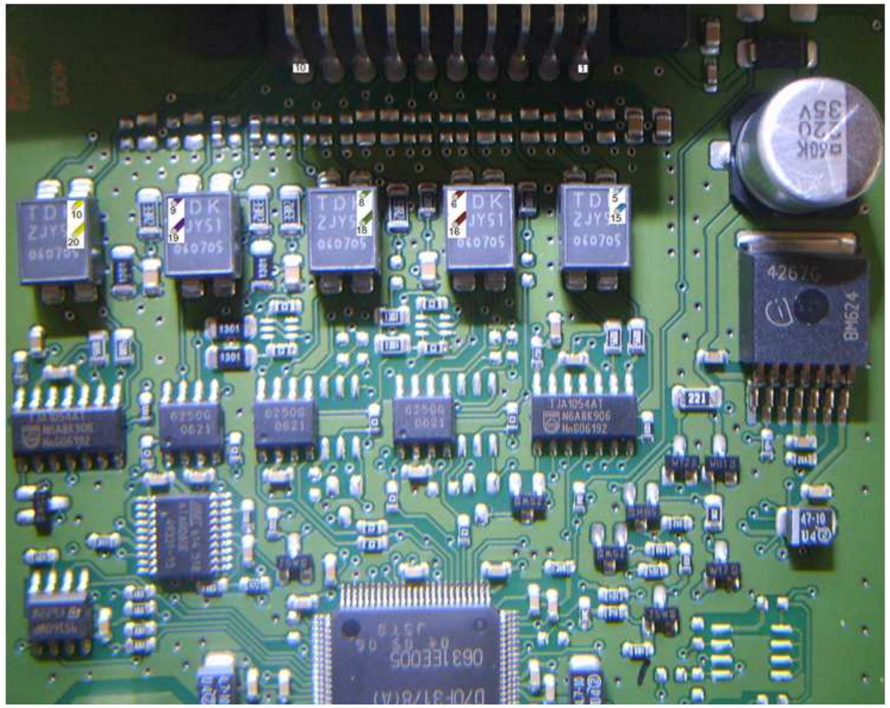
SIEMENS VDO 0186 GATEWAY
02 10.09.06
A2C53094005 VP1 - PIR 02 Made in Malaysia

pin1,pin2: +12V (car-battery); both inside wired

pin11,pin12: GND (car-ground) ; both inside wired

pin14: on/off; Gateway on(+12V) and

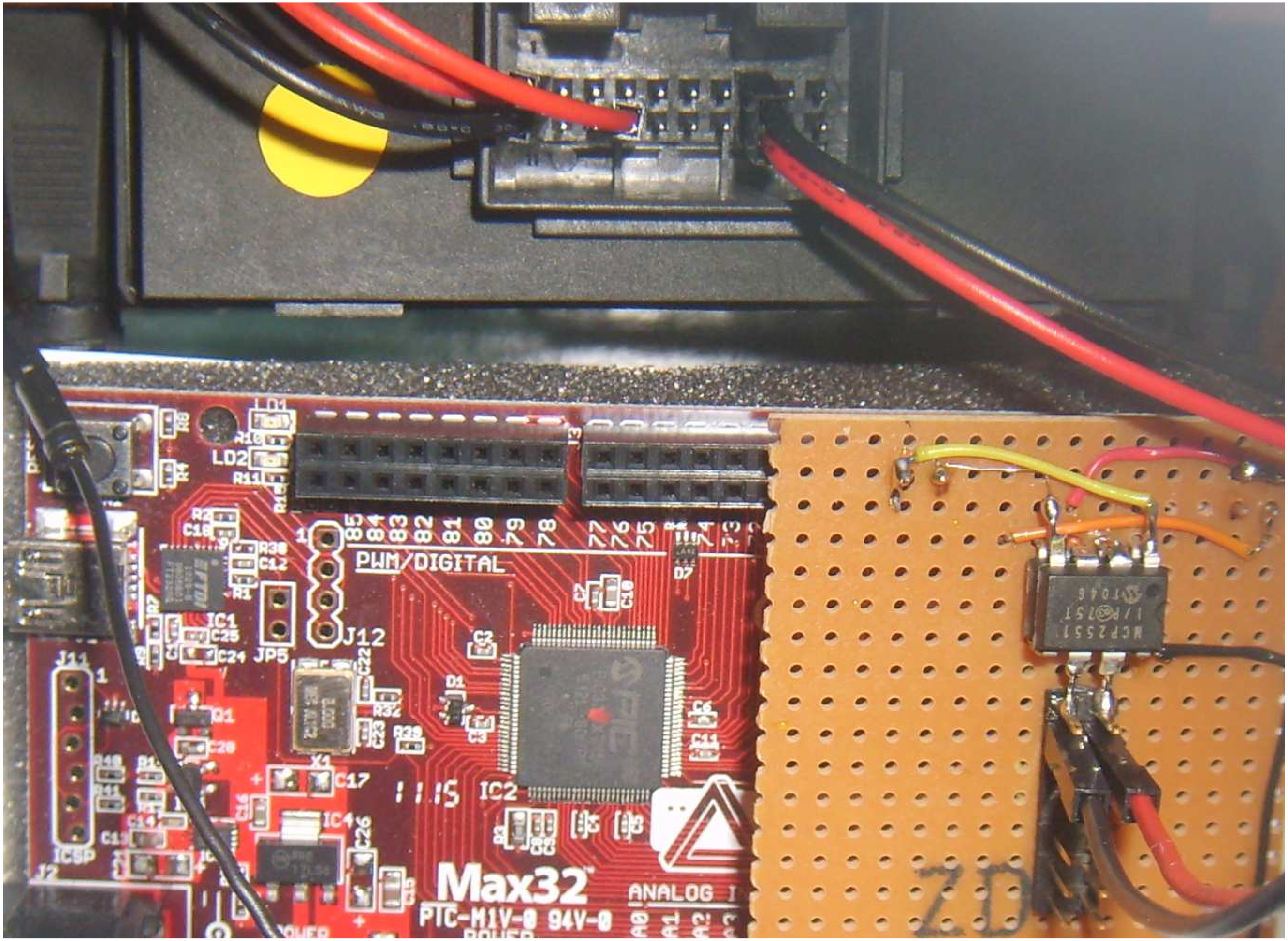
Gateway off(GND or not connected)



We use:

8 pin8, CAN_L, 500kbps

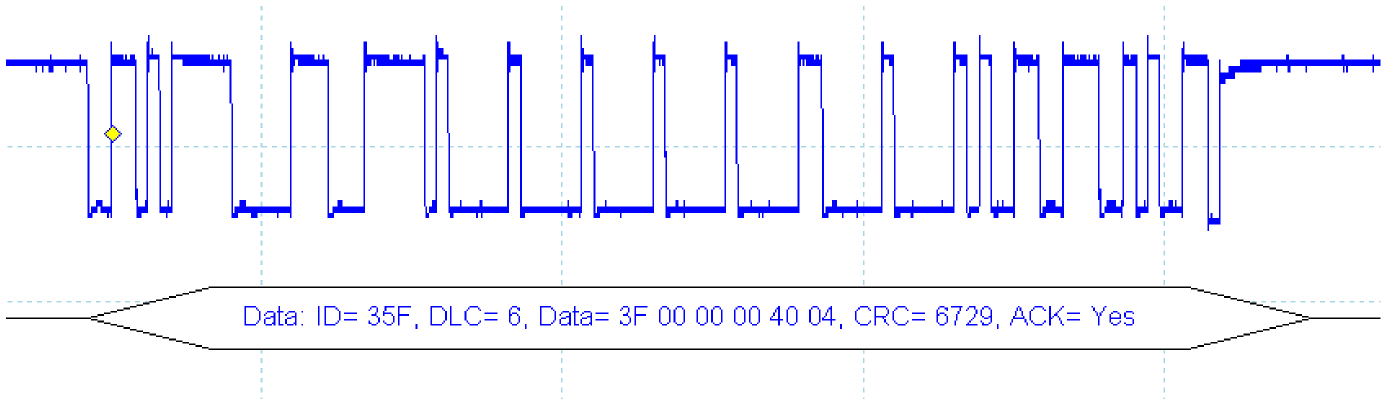
18 pin18, CAN_H

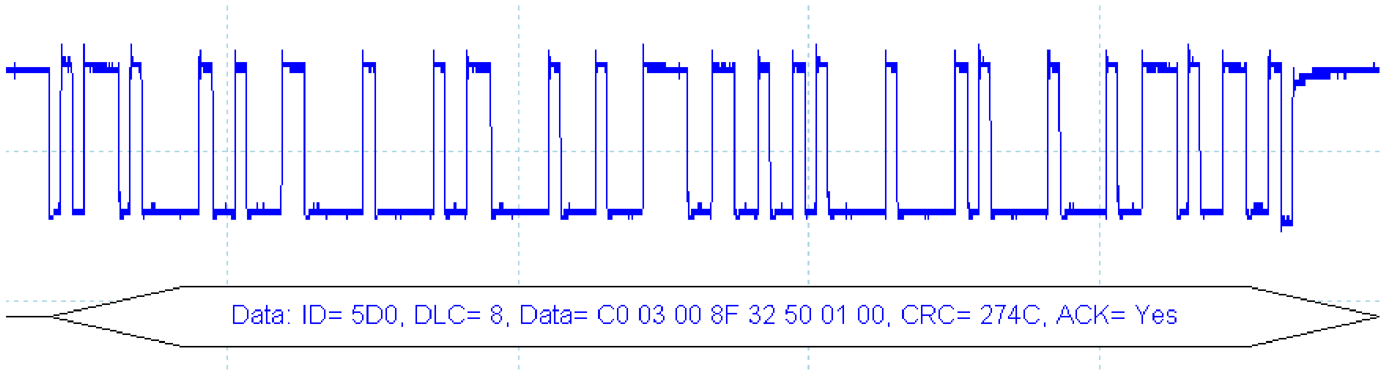

PWM/DIGITAL
J12
JP5
IC1
IC2
ICSP
Max32
PTC-M1V-0 94V-0
ANALOG I
MCP2551
ZD

## Simple monitor:

```
#include  <WProgram.h>
#include  "chipKITCAN.h"

#define SYS_FREQ  (80000000L)
#define CAN_BUS_SPEED   500000       // CAN Speed

int data[8];
CAN    canMod1(CAN::CAN1);    // this object uses CAN module 1
uint8_t  CAN1MessageFifoArea[2 * 8 * 16];
uint16_t id_0; //or uint32_t
static volatile bool isCAN1MsgReceived = false;

void initCan1();
void rxCAN1(void);
void doCan1Interrupt();

void setup() {
  initCan1();
  canMod1.attachInterrupt(doCan1Interrupt);
  Serial.begin(9600);
}

void loop() {
  rxCAN1();
}

void initCan1() {
  CAN::BIT_CONFIG canBitConfig;
  canMod1.enableModule(TRUE);
  canMod1.setOperatingMode(CAN::CONFIGURATION);
  while(canMod1.getOperatingMode() != CAN::CONFIGURATION);
  canBitConfig.phaseSeg2Tq            = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq            = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect    = TRUE;
  canBitConfig.sample3Time            = TRUE;
  canBitConfig.syncJumpWidth          = CAN::BIT_2TQ;
  canMod1.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
  canMod1.assignMemoryBuffer(CAN1MessageFifoArea,2 * 8 * 16);
  canMod1.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE); //RX_DATA_ONLY
  canMod1.configureFilterMask  (CAN::FILTER_MASK0, 0x0, CAN::SID, CAN::FILTER_MASK_ANY_TYPE);//CAN_FILTER_MASK_IDE_TYPE
  canMod1.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod1.enableFilter         (CAN::FILTER0, TRUE);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
  canMod1.enableModuleEvent(CAN::RX_EVENT, TRUE);
  canMod1.setOperatingMode(CAN::NORMAL_OPERATION);
  while(canMod1.getOperatingMode() != CAN::NORMAL_OPERATION);
}

void rxCAN1(void) {
  int k;
  CAN::RxMessageBuffer *message;
  if (isCAN1MsgReceived == false) {
    return;
  }
  isCAN1MsgReceived = false;
  message = canMod1.getRxMessage(CAN::CHANNEL1);
  id_0 = message->msgSID.SID;
  Serial.print(id_0, HEX);
  Serial.print(", ");
  for(k=0;k<8;k++)
    data[k] = message->data[k];
  for(k=0;k<7;k++){
    Serial.print(data[k], HEX);
    Serial.print(", ");
  }
  Serial.println(data[7], HEX);
  canMod1.updateChannel(CAN::CHANNEL1);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, TRUE);
}

void doCan1Interrupt() {
  if ((canMod1.getModuleEvent() & CAN::RX_EVENT) != 0) {
    if(canMod1.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
      canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, FALSE);
      isCAN1MsgReceived = true;
    }
  }
}

/*

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0
```

```
657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

557, 0, 0, 0, 0, 8, 0, 0, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

557, 0, 0, 0, 0, 8, 0, 0, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

35F, 3F, 0, 0, 0, 40, 4, 0, 0

5D0, C0, 3, 0, 8F, 32, 50, 1, 0

657, 0, 0, 80, 0, 8, 0, 0, 0

*/
```





… have fun! edgarmarx@t-online.de