NXP’s ARM7 (LPC2148), ARM Slicker Kit is proposed to smooth the progress of developing and debugging of various designs encompassing of High speed 32-bit Microcontrollers.

**SPI (Serial Peripheral Interface)**

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

**CAN (Controller Area Network)**

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs). Each node is able to send and receive messages, but not simultaneously. CAN bus is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. The CAN communication protocol is a CSMA/CD protocol.

**The MCP2510**

The MCP2510 is a Full CAN protocol controller implementing CAN specification V2.0 A/B. It is capable of transmitting & receiving standard and extended messages. It is also capable of both acceptance filtering and message management. It includes three transmit buffers and two receive buffers that reduce the amount of microcontroller management required. The MCU communication is implemented via SPI with data rates up to 5 Mb/s.

**Interfacing SPI-CAN**

Fig. 1 shows how to interface the SPI-CAN to microcontroller. With an SPI connection there is always one master device which controls the peripheral devices. Typically there are three lines common to all the devices,

* Master In Slave Out (MISO) - The Slave line for sending data to the master,
* Master Out Slave In (MOSI) - The Master line for sending data to the peripherals,
* Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master&
* Slave Select pin - the pin on each device that the master can use to enable and disable specific devices. When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master.

This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

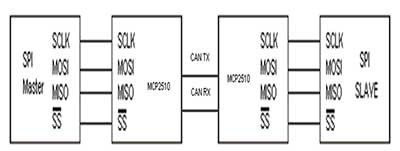


Fig. 1 Interfacing SPI-CAN to Microcontroller

The CAN protocol engine handles all functions for receiving and transmitting messages on the bus. Messages are transmitted by first loading the appropriate message buffer and control registers. The MCP2510 has three transmit and two receive buffers, two acceptance masks (one for each receive buffer), and a total of six acceptance filters. The CRC Register generates the Cyclic Redundancy Check code which is transmitted after either the Control Field (for messages with 0 data bytes) or the Data Field, and is used to check the CRC field of incoming messages.

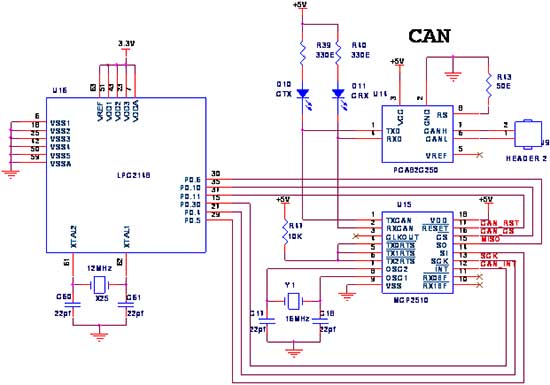
**Interfacing SPI-CAN with LPC2148**

In SPI, the clock signal is controlled by the master device LPC2148 Slicker Board. All data is clocked in and out using this pin. These lines need to be connected to the relevant pins on the LPC2148 Slicker Board. In Mcp2510, Transmission is initiated by using control register bits, via the SPI interface, or by using the transmit enable pins. Interrupt pins are provided to allow greater system flexibility. There are also three pins available to initiate immediate transmission of a message that has been loaded into one of the three transmit registers.

**Pin Assignment with LPC2148**

|  |  |
| --- | --- |
| **CAN - SPI Connector** | **Processor Lines** |
| SCK | P0.4 |
| MISO | P0.6 |
| RST | P0.11 |
| CS | P0.10 |
| INT | P0.30 |

**Circuit Diagram to Interface SPI-CAN with LPC2148**



**Source Code**

The Interfacing SPI-CAN with LPC2148 program is very simple and straight forward, which controls the LPC2148 Slicker Board from another LPC2148 Slicker Board using CAN via SPI. Some delay is occurring when a single data is sent through SPI. C programs are written in Keil software.

**C Program to interface SPI-CAN with LPC2148**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Title : Program for SPI-Master

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <LPC214X.H>

#include "CAN\_SPIsw.h"

#include "CAN\_SPIconst.h"

#include "SPIsw.h"

#include "Utility.h"

unsigned char aa,aa1,aa2,len,zr,cont,oldstate,data[8];

unsigned long id;

int main (void)

{

   PINSEL0 = 0;

   PINSEL1 = 0;

   PINSEL2 &= 0x0000000C;

   //--- PORT1[23..16] as Initialization pgr indicator

   IOCLR1 |= ((unsigned long)0xFF << 16);

   IODIR1 |= ((unsigned long)0xFF << 16);

   //--- PORT1[31..24] as Data Value indicator

   IOCLR1 |= ((unsigned long)0xFF << 24);

   IODIR1 |= ((unsigned long)0xFF << 24);

   //--- initialize SPI (mode 0,0)

   IODIR0 |= ((unsigned long)1 << 11); // P0.11 output

   IODIR0 |= ((unsigned long)1 << 10); // P0.10 output

   IOSET0 |= ((unsigned long)1 << 11); // RESET line

   IOCLR0 |= ((unsigned long)1 << 10); // CS line

   SPI\_init(&IOPIN0, 10/\*CS\*/, 5/\*MISO\*/, 6/\*MOSI\*/,

    4/\*SCK\*/, 0/\*CPOL\*/, 0/\*CPHA\*/);   //--- configure CAN\_SPI module

   aa = 0;

   aa1 = 0;

   aa2 = 0;

   aa =   CAN\_CONFIG\_SAMPLE\_THRICE &

            CAN\_CONFIG\_PHSEG2\_PRG\_ON &

            CAN\_CONFIG\_STD\_MSG &

            CAN\_CONFIG\_DBL\_BUFFER\_ON &

            CAN\_CONFIG\_VALID\_XTD\_MSG;

   aa1 =   CAN\_TX\_PRIORITY\_0 &

            CAN\_TX\_XTD\_FRAME &

            CAN\_TX\_NO\_RTR\_FRAME;

   IOSET0 |= (1 << 10);

  // set CS line for CAN module high (inactive)

   id = -1;

  // id temporarily used for SetMask operation

   CANSPIInitialize(1,1,3,3,1,aa);

 // initialize external CAN module

   IOCLR1 |= (0xFF << 16);  // PORT1[23..16] = 3;

   IOSET1 |= (0x03 << 16);

   CANSPISetOperationMode(CAN\_MODE\_CONFIG,0xFF);

   IOCLR1 |= (0xFF << 16); // PORT1[23..16] = 4;

   IOSET1 |= (0x04 << 16);

CANSPISetMask(CAN\_MASK\_B1,id,CAN\_CONFIG\_XTD\_MSG);            // set all mask1 bits to ones

CANSPISetMask(CAN\_MASK\_B2,id,CAN\_CONFIG\_XTD\_MSG);            // set all mask2 bits to ones

CANSPISetFilter(CAN\_FILTER\_B2\_F4,3,CAN\_CONFIG\_XTD\_MSG);         // set id of filter B1\_F1 to 3

CANSPISetOperationMode(CAN\_MODE\_NORMAL,0xFF);                        // set NORMAL mode

   IOCLR1 |= (0xFF << 16);  // PORT1[23..16] = 5;

   IOSET1 |= (0x05 << 16);

   data[0] = 7;

   id = 12111;

   IOCLR1 |= (0xFF << 16); // PORT1[23..16] = 255;

   IOSET1 |= (0xFF << 16);

   CANSPIWrite (id,data,1,aa1); / 1st node communication

   while (1)

   {

      zr = CANSPIRead(&id , data , &len, &aa2);

      if ((id == 3) && zr)

      // respond only to data from node No.3

      {

         IOCLR1 |= (0xFF << 16);

         IOSET1 |= ((unsigned long)data[0] << 16) & 0x00FF0000;

         data[0]++ ;

         id = 12111;

         delay\_Nx10cyc(5999999);      // Delay 1s

         CANSPIWrite(id, data, 1,aa1);

      }

   }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Title : Program for SPI-Slave

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <LPC214X.H>

#include "CAN\_SPIsw.h"

#include "CAN\_SPIconst.h"

#include "SPIsw.h"

#include "Utility.h"

unsigned char aa,aa1,aa2,len,zr,cont,oldstate,data[8];

unsigned long id;

int main (void)

{

   PINSEL0 = 0;

   PINSEL1 = 0;

   PINSEL2 &= 0x0000000C;

 //--- PORT1[23..16] as Initialization progress indicator

   IOCLR1 |= ((unsigned long)0xFF << 16);

   IODIR1 |= ((unsigned long)0xFF << 16);

 //--- prepare port PORT1[31..24] as Data Value indicator

   IOCLR1 |= ((unsigned long)0xFF << 24);

   IODIR1 |= ((unsigned long)0xFF << 24);

   //--- initialize SPI (mode 0,0)

   IODIR0 |= ((unsigned long)1 << 11);  // P0.11 output

   IODIR0 |= ((unsigned long)1 << 10);  // P0.10 output

   IOSET0 |= ((unsigned long)1 << 11);  // RESET line

   IOCLR0 |= ((unsigned long)1 << 10);  // CS line

   SPI\_init(&IOPIN0, 10/\*CS\*/, 5/\*MISO\*/, 6/\*MOSI\*/,

 4/\*SCK\*/, 0/\*CPOL\*/, 0/\*CPHA\*/);   //--- configure CAN\_SPI module

   aa = 0;

   aa1 = 0;

   aa2 = 0;

   aa =   CAN\_CONFIG\_SAMPLE\_THRICE &

            CAN\_CONFIG\_PHSEG2\_PRG\_ON &

            CAN\_CONFIG\_STD\_MSG &

            CAN\_CONFIG\_DBL\_BUFFER\_ON &

            CAN\_CONFIG\_VALID\_XTD\_MSG;

   aa1 =   CAN\_TX\_PRIORITY\_0 &   // form value to be used

            CAN\_TX\_XTD\_FRAME &   // with CANSPIWrite

            CAN\_TX\_NO\_RTR\_FRAME;

   IOSET0 |= (1 << 10); // set CS line

   id = -1; // id temporarily used for SetMask operation

   CANSPIInitialize(1,1,3,3,1,aa);  // external CAN module

   IOCLR1 |= (0xFF << 16);         // PORT1[23..16] = 3;

   IOSET1 |= (0x03 << 16);

   CANSPISetOperationMode(CAN\_MODE\_CONFIG,0xFF);

   IOCLR1 |= (0xFF << 16); // PORT1[23..16] = 4;

   IOSET1 |= (0x04 << 16);

CANSPISetMask(CAN\_MASK\_B1,id,CAN\_CONFIG\_XTD\_MSG);                     CANSPISetMask(CAN\_MASK\_B2,id,CAN\_CONFIG\_XTD\_MSG);                  // set all mask2 bits to ones

CANSPISetFilter(CAN\_FILTER\_B2\_F4,12111,CAN\_CONFIG\_XTD\_MSG);         // set id of filter B1\_F1 to 12111

CANSPISetOperationMode(CAN\_MODE\_NORMAL,0xFF);                        // set NORMAL mode

   IOCLR1 |= (0xFF << 16);  // PORT1[23..16] = 5;

   IOSET1 |= (0x05 << 16);

   data[0] = 9;

   data[1] = 0x81;

   id = 3;

   IOCLR1 |= (0xFF << 16);  // PORT1[23..16] = 255;

   IOSET1 |= (0xFF << 16);

   while (1)

   {

      zr = CANSPIRead(&id , data , &len, &aa2);

      if ((id == 12111) && zr)

      // respond only to data from node No.12111

      {

         IOCLR1 |= (0xFF << 16); // PORT1[23..16]

         IOSET1 |= ((unsigned long)data[0] << 16) & 0x00FF0000;

         data[0]++ ;

         id = 3;  // id contains CAN network ID of this node

         CANSPIWrite(id, data,8,aa1);

      }

   }

}

To compile the above C code you must need the **KEIL software.** They must be properly set up and a project with correct settings must be created in order to compile the code. To compile the above code,**the C**file must be added to the project.

In KEIL, you want to develop or debug the project without any hardware setup. You must compile the code for generating HEX file. In debugging Mode, you want to check the port output without LPC2148 Slicker Board.

The Flash Magic software is used to download the hex file into your microcontroller IC LPC2148 through UART0.

**Testing the SPI-CAN with LPC2148**

Give +3.3V power supply to LPC2148 Slicker Board; the SPI-CAN device is connected with LPC2148 Slicker Board. Here we are using Master and Slave LPC2148 Slicker Board for CAN transmit & receive.

When the Master Board receives any signals from Slave then it send data to Slave. If the Slave Board receives data from Master Board then it can enables some LED’s for indication. Then the Slave sends some acknowledgeable data into Master LPC2148 Slicker Board. Now the Master Board enables some LED for indication.

If you are not reading any output from LED, then you just check the jumper connections & check the LED is working. Check the CAN device is working or not & check the connections of Master & Slave LPC2148 Slicker Board.

Otherwise you just check the code with debugging mode in KEIL. If you want to see more details about debugging just see the videos in below link.

- See more at: https://www.pantechsolutions.net/microcontroller-boards/spi-can-interfacing-with-lpc2148-arm7-slicker#sthash.2RD2kIrD.dpuf