Smart Card Library

Table of Contents

Smart Card Library

1 Smart Card Library

1.1 Introduction

Microchip Smart Card Library Help Documentation

Description

Smart Card EMV Standard Library **For 16 bit PIC Microcontrollers**

The Smart Card library for PIC microcontrollers support EMV Level 1 based on ISO 7816-3 standard . It allows the PIC microcontroller to communicate with smart card's compatible with these protocols. The library supports both T=0 and T=1 smart card protocols.

The library comprises of PIC24 UART driver and T0/T1 protocol source code meeting Smart Card EMV and ISO-7816-3 standards. An example high level demo application code is also provided to help the user port the smart card library to different microcontrollers of PIC family (PIC24FJ128GB204/GA204 and PIC24FJ256GB110 by using normal UART).

This document assumes that the reader is familiar with Smart Card EMV/ISO 7816-3 standards and T=0/T=1 protocols.

1.2 Legal Information

This software distribution is controlled by the Legal Information at www.microchip.com/mla_license

1.3 Release Notes

1.3.1 v2.00

The v2.00 release is a major releases with the following changes.

1. File names, Function names, Macro names and structure names have been changed

2. T=1 protocol support has been added

3. T=0 and T=1 are run time selectable after every "Answer To Reset"

1.3.2 v1.02.6

1. In SClib.c.:-
   - Changed the size of input/output parameters of static functions 'SC_UpdateCRC', 'SC_UpdateEDC' and 'SC_SendT1Block'. This fix is done to optimize the code.
   - Modified the contents of 'SC_UpdateCRC' and 'SC_SendT1Block' function to suit the above change.
   - Modified "SC_TransactT0" function, to transmit first byte as 0x00 when LC and LE bytes are 0x00.
   - Changed the local variable 'edc' from 'WORD' type to 'unsigned short int' type (in static function :- 'SC_ReceiveT1Block')
2. In SCpic24.c, SCpic18.c, SCpic32.c and SCdspic33f.c:-
   - The variable 'delayLapsedFlag' is declared as 'volatile' type, as it is modified in the Interrupt Service Routine.

1.3.3 v1.02.4

1. In SClib.c.:-
   - The wait time was getting reinitialized to default value while communicating with smart card using T = 0 protocol. So deleted "t0WWTetu = 10752;" in "SC_TransactT0" function.
   - Modified the function "SC_SendT1Block" in such a way that EDC is transmitted more effeciently for LRC/CRC mode in T = 1 protocol.
   - Initialized local variable "txLength" to '0' in function "SC_TransactT1" to remove non-critical compiler warnings.
2. In sc_config.h
   - Removed the following unused file inclusions:-
     1. libpic30.h
     2. math.h
     3. delays.h
     4. plib.h

1.3.4 v1.02.2

1. Modified the PPS functionality as per ISO 7816 standard.
2. Fixed BWT (Block Wait Time) and WT (Wait Time) calculation issues.
3. Removed recursive function calls and modified the code to make it well structured and organized.
4. Modified "SCdrv_EnableDelayTimerIntr" and "SCdrv_SetDelayTimerCnt" macros to configure 16 bit timers (this macro is used to provide delays).
5. “WaitMicroSec()" and "WaitMilliSec()" macros are removed from sc_config.h file.
6. Moved timer interrupts (used by smart card stack) to ISO 7816 hardware driver files.
7. Added "TIMER1_SINGLE_COUNT_MICRO_SECONDS" and "TIMER0_SINGLE_COUNT_MICRO_SECONDS" macros in sc_config.h file.
8. WaitMicroSec() and WaitMilliSec() delay functions have been rewritten in the ISO 7816 driver files to provide accurate delays.
9. The following PPS response variables have been added as part of the global memory.

Names	Description
scPPSresponse[7]	PPS Response Bytes from smart card
scPPSresponseLength	Length of PPS Response

The prototype definition of function “SC_DoPPS( )” has been changed to “SC_DoPPS(BYTE *ppsPtr)”. The input parameter for “SC_DoPPS” function is PPS request string. This feature enables the user to send the desired PPS request to the card.

1.3.5 v1.02

Supported smart card library stack to PIC32, PIC24H and dsPIC33F devices.

1.3.6 v1.01

The following list of variable names has been changed to follow a common coding standard across the smartcard library.

Changed From	Changed To
SC_CardATR	scCardATR
SC_ATRLen	scATRLength
SC_LastError	scLastError
SC_TA1	scTA1
SC_TA2	scTA2
SC_TA3	scTA3
SC_TB1	scTB1
SC_TB2	scTB2
SC_TB3	scTB3
SC_TC1	scTC1

SC_TC2	scTC2
SC_TC3	scTC3
SC_TD1	scTD1
SC_TD2	scTD2
SC_TD3	scTD3
SC_ATR_HistBfr	scATR_HistoryBuffer
SC_ATR_HistLen	scATR_HistoryLength

The following list of type definitions has been changed to make them more understandable.

Changed From	**Changed To**
SC_APDU_Cmd	SC_APDU_COMMAND
SC_APDU_Resp	SC_APDU_RESPONSE

The function name “SC_Transact” has been changed to “SC_TransactT0” to signify that this function handles only T=0 transactions with the smart card.

The function name “SC_TransactT1” has been added newly to signify that this function handles only T=1 transactions with the smart card. The application has to call “SC_TransactT0” or “SC_TransactT1” function depending upon the card inserted.

1.3.7 v1.02.8

1. In SClib.c.:-
    - "SC_TransactT0" function is modified to handle a 256 bytes read from smart card as per the "Case 2S" requirement of ISO 7816 specification.
    - The assignment of "apduResponse->SW1" and "apduResponse->SW2" is modified in "SC_TransactT1" function
2. In SCpic24.c, SCpic18.c, SCpic32.c and SCdspic33f.c:-
    - "SCdrv_InitUART" function is modifed to switch on the power supply to the smart card during initialization phase.

1.3.8 v1.03

1. In SClib.c.:-
    - Changed the data type of variable "cgtETU" from "BYTE" to "unsigned short int".
    - Modified "SC_DoPPS" function, so as to add the guard time between transmission of bytes to smart card.
    - Modified "SC_CalculateWaitTime" function, so as to calculate correct guard and wait time values.

1.4 Using the Library

1.4.1 Smart Card Library Overview

Two communication protocols that are generally used for contact type Smart Card communications are:

- T = 0 (asynchronous half duplex character transmission)
- T = 1 (asynchronous half duplex block transmission)

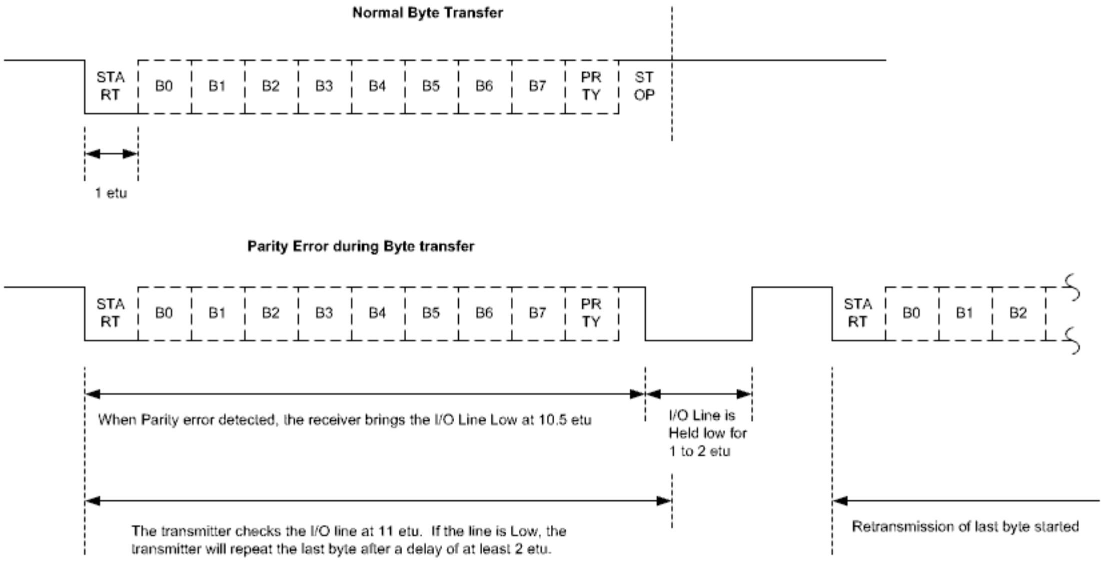
The data transfers between the card and the terminal(smart card reader) happens on the single wire I/O line.

Following the initial reset of the card after insertion, the card responds with a series of characters called the Answer to Reset, or ATR. This series of characters establishes the initial communication details, including the specific protocol, bit timing, and data transfer details for all subsequent communications. While subsequent data transfers can change certain communications parameters, the ATR establishes initial communications conditions.

The Clock Signal for Baud rate generation is provided to the card by the reader (terminal). The Smart Card default baudrate divider is 372, which produce 9600 bps when a clock signal of 3.57MHz is supplied to the card. Most Smart Cards allow higher clock rates, so a simple 4MHz clock can be easily used. Using a 4MHz clock, the default baudrate comes out to be 10752 bps. The PICs UART is appropriately configured by the library, so the communication can be setup using the higher baudrate settings.

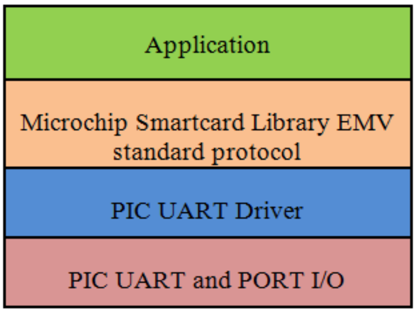
The Smart Card 7816-3 communications requires a 0.5 stop bit. This is important for the Receiver, as it must pull the I/O line low before the middle of the stop bit (10.5 bit time from start edge) in order to indicate error condition to the Transmitter. The receiver pulls the line low for 1 to 2 bit time (etu). The transmitter checks the I/O line at the end of stop bit, or 11 etu. If the transmitter detects the line low, it retransmit the previous data byte after at least 2 etu.

The uart peripheral in PIC micros sets Rx Ready and Transmitter Empty flags to true at 0.5 stop bit, which allows the implementation of the 7816-3 error detection and retransmission protocol possible.


Normal Byte Transfer
STA RT
B0
B1
B2
B3
B4
B5
B6
B7
PR TY
ST OP
1 etu
Parity Error during Byte transfer
STA RT
B0
B1
B2
B3
B4
B5
B6
B7
PR TY
STA RT
B0
B1
B2
When Parity error detected, the receiver brings the I/O Line Low at 10.5 etu
I/O Line is Held low for 1 to 2 etu
The transmitter checks the I/O line at 11 etu. If the line is Low, the transmitter will repeat the last byte after a delay of at least 2 etu.
Retransmission of last byte started

1.4.2 Library Architecture

The Smart Card Library has a modular design with separate files for the high level library code and the low level driver for UART for implementing the EMV/ISO 7816-3 standards of Smart Card protocol.



1.4.3 How the Library Works

The current release of Smart Card library supports PIC24F microcontrollers.The Smart Card library provides the API necessary to communicate with the EMV standard compliant Smart Card. The sequence of the API calls is as given below. smart_card.h contains all the API's that are required by the main application to communicate with the Smart Card. The current release of smart card library supports both T=0 and T=1 protocol.

```
...

//Initialize smart card stack

SMARTCARD_Initialize();

...

// Wait untill the card is inserted in the slot

while( !SMARTCARD_IsPresent() )

...

//After detecting the card, turn on the power to the card and process Answer-to-Reset

if( !SMARTCARD_PowerOnATR() )

...

//Do protocol and parameter selection.Configure the desired baud rate

if( SMARTCARD_IsPPSSupported() )

...

//Execute Card Commands

//If card is inserted in the slot, execute T=1/T=0 based on ATR commands

if(SMARTCARD_ProtocolTypeGet)

{

//If T=0 card is inserted in the slot, execute T=0 commands

if( !SMARTCARD_EMV_DataExchangeT0( &cardCommand,apduCommandLength,&demo_CardResponse ))
```

```
}
else if(!SMARTCARD_EMV_DataExchangeT0( &cardCommand,apduCommandLength,&demo_CardResponse ))
{
...
...
// Shut Down the Card when there is nothing to do with it
SMARTCARD_Shutdown;
...
```

Note :

1)For T=1 protocol "prologueField" buffer should contain the prologue field(NAD,PCB,LENGTH) that needs to be sent to Smart Card.Once the transaction is completed between the card and the micro, response from the card is stored in "cardResponse" buffer."apduData" points to the data buffer of the command as well as data response from the card.

2)For T=0 protocol "cardCommand" buffer should contain the command that needs to be sent to the Smart Card. Once the transaction is completed between the card and the micro, response from the card is stored in "cardResponse" buffer. "apduData" points to the data buffer of command as well as data response from the card.

1.4.4 Integrating with an Existing Application

It is easy to integrate the Smart Card library with the existing applications.The Smart Card library uses UART and 4 I/O port pins.

The pins used for the communication between the Smart Card and PIC microcontroller are given in Configuring the Library section.“smart_card_config.h” is the only file where the user has to modify to port the Smart Card stack to different PIC microcontrollers.

The API's that needs to be called by the main application are mentioned in smart_card.h file.Please refer "How the Library Works" to know the usage of smart card library API's.

1.5 Library Interface

1.5.1 Functions

1.5.1.1 Initialization Functions

Functions

	Name	Description
◆	SMARTCARD_Initialize	This function initializes the smart card library
◆	SMARTCARD_PowerOnATR	This function performs the power on sequence of the SmartCard and interprets the Answer-to-Reset data received from the card.

1.5.1.1.1 SMARTCARD_Initialize Function

File

smart_card.h

Syntax

```
void SMARTCARD_Initialize();
```

Description

This function initializes the smart card library

Remarks

None

Preconditions

None

Function

void SMARTCARD_Initialize(void)

1.5.1.1.2 SMARTCARD_PowerOnATR Function

File

smart_card.h

Syntax

```
SMARTCARD_TRANSACTION_STATUS SMARTCARD_PowerOnATR(SMARTCARD_RESET_TYPES resetRequest);
```

Description

This function performs the power on sequence of the SmartCard and interprets the Answer-to-Reset data received from the card.

Remarks

None

Preconditions

SMARTCARD_Initialize() is called, and card is present

Example

```
 {

    while(!SMARTCARD_IsPresent());
            ...
            ...
            contact = TypeOfCard();
    if(contact)
            {
                    return(SMARTCARD_PowerOnATR); //Contact based Smart card
            }
            else
            {
                    return(return(SCL_PowerOnATR););  //Contact-less based Smart Card
            }

}
```

Function

SMARTCARD_TRANSACTION_STATUS SMARTCARD_PowerOnATR(SMARTCARD_RESET_TYPE resetRequest)

1.5.1.2 Transaction Functions

Functions

	Name	Description
◆	SMARTCARD_DataExchange	This function performs the data transaction, by calling the appropriate routine based upon card type.
◆	SMARTCARD_EMV_ATRProcess	This function performs the power on sequence of the SmartCard and interprets the Answer-to-Reset data received from the card.
◆	SMARTCARD_EMV_DataExchangeT0	This function Sends/receives the ISO 7816-4 compliant T = 0 commands to the card.
◆	SMARTCARD_EMV_DataExchangeT1	This function Sends/receives the ISO 7816-4 compliant T = 1 commands to the card.
◆	SMARTCARD_IsPPSSupported	This function gets whether PPS(Protocol & Parameter Selection) is supported or not
◆	SMARTCARD_IsPresent	This macro checks if card is inserted in the socket
◆	SMARTCARD_PPSExchange	This function does the PPS exchange with the smart card & configures the baud rate of the PIC UART module as per the PPS response from the smart card.
◆	SMARTCARD_ProtocolTypeGet	This function gets the type of the protocol supported by the card.
◆	SMARTCARD_Shutdown	This function Performs the Power Down sequence of the SmartCard
◆	SMARTCARD_StateGet	This function returns the current state of SmartCard

1.5.1.2.1 SMARTCARD_DataExchange Function

File

smart_card.h

Syntax

```
uint8_t SMARTCARD_DataExchange(SMARTCARD_APDU_COMMAND* apduCommand);
```

Description

This function performs the data transaction, by calling the appropriate routine based upon card type.

Remarks

None

Preconditions

SMARTCARD_Initialize() is called, and card is present

Example

```
 {

    while(!SMARTCARD_IsPresent());
            ...
            ...
            contact = TypeOfCard();
    if(contact)
            {
                    SMARTCARD_EMV_T0();
            }
            else
            {
                    ...
            }

    }
```

Function

uint8_t SMARTCARD_DataExchange(SMARTCARD_APDU_COMMAND* apduCommand)

1.5.1.2.2 SMARTCARD_EMV_ATRProcess Function

File

smart_card_layer3.h

Syntax

```
SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_ATRProcess(SMARTCARD_RESET_TYPES resetRequest);
```

Description

This function performs the power on sequence of the SmartCard and interprets the Answer-to-Reset data received from the card.

Remarks

None

Preconditions

SMARTCARD_Initialize() is called, and card is present

Parameters

Parameters	Description
SMARTCARD_RESET_TYPES resetRequest	type of reset requested by the card

Function

SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_ATRProcess(SMARTCARD_RESET_TYPE resetRequest)

1.5.1.2.3 SMARTCARD_EMV_DataExchangeT0 Function

File

smart_card_layer3.h

Syntax

```
SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_DataExchangeT0(uint8_t* apduCommand, uint32_t
apduCommandLength, SMARTCARD_APDU_RESPONSE* apduResponse);
```

Description

This function Sends/receives the ISO 7816-4 compliant T = 0 commands to the card.

Remarks

None

Preconditions

SMARTCARD_PPS was success

Example

```
main()
{

    while(!SMARTCARD_IsPresent());
    scError = LoopBackMode(transType);
    ....
    ....
    // Send Command APDU and get Response APDU
     EMV_APDU (transactionType); // trascationType=T0/T1
     ....
     if(transactionType == T0_TYPE)
    {
            return(SMARTCARD_EMV_DataExchangeT0(&apduCmd[0], apduCmdLength, &cardResponse));
    }
    else
    {
            return(SMARTCARD_EMV_DataExchangeT1(&pField,&apduCmd[0],&cardResponse));
//SMARTCARD_EMV_TransactT1
    }

}
```

Function

SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_DataExchangeT0(uint8_t* apduCommand, uint32_t apduCommandLength, SMARTCARD_APDU_RESPONSE* apduResponse)

1.5.1.2.4 SMARTCARD_EMV_DataExchangeT1 Function

File

smart_card_layer3.h

Syntax

```
SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_DataExchangeT1(SMARTCARD_T1PROLOGUE_FIELD*
pfield, uint8_t* iField, SMARTCARD_APDU_RESPONSE* apduResponse);
```

Description

This function Sends/receives the ISO 7816-4 compliant T = 1 commands to the card.

Remarks

None

Preconditions

SC_PPS was success

Example

Refer to SMARTCARD_EMV_DataExchangeT0() function

Function

SMARTCARD_TRANSACTION_STATUS SMARTCARD_EMV_DataExchangeT1(SMARTCARD_T1_PROLOGUE_FIELD* pfield,uint8_t* iField,SMARTCARD_APDU_RESPONSE* apduResponse)

1.5.1.2.5 SMARTCARD_IsPPSSupported Function

File

smart_card.h

Syntax

```
SMARTCARD_PPS_SUPPORT_STATUS SMARTCARD_IsPPSSupported();
```

Description

This function gets whether PPS(Protocol & Parameter Selection) is supported or not

Remarks

None

Preconditions

SMARTCARD_Initialize is called and CARD is in ATR on state.

Example

```
statement1;
 if (resetRequest == WARM_RESET)
 {
     SMARTCARD_Shutdown();
 }
 ...
 ...
 ...

 SMARTCARD_PowerOnATR();
 ...
 ...
 // Return False if there is no card inserted in the Slot or ATR of the card is unsuccessful
 if( !SCdrv_CardPresent() || (gCardState != ATR_ON ))
 {
     SMARTCARD_Shutdown();
     return SC_ERR_CARD_NOT_PRESENT;
 }
```

Function

void SMARTCARD_IsPPSSupported(void)

1.5.1.2.6 SMARTCARD_IsPresent Function

File

smart_card.h

Syntax

```
bool SMARTCARD_IsPresent();
```

Description

This macro checks if card is inserted in the socket

Remarks

None

Preconditions

SMARTCARD_Initialize() is called

Example

```
main())
{
    SMARTCARD_Initialize();
    while(!SMARTCARD_IsPresent());
    // Do other tasks only when the card is detected
}
```

Function

bool SMARTCARD_IsPresent(void)

1.5.1.2.7 SMARTCARD_PPSExchange Function

File

smart_card_layer3.h

Syntax

```
SMARTCARD_TRANSACTION_STATUS SMARTCARD_PPSExchange(uint8_t * ppsPtr);
```

Description

This function does the PPS exchange with the smart card & configures the baud rate of the PIC UART module as per the PPS response from the smart card.

Remarks

This function is called when SMARTCARD_EMV_ATRProcess() returns 1.

Preconditions

SMARTCARD_PowerOnATR was success

Function

SMARTCARD_TRANSACTION_STATUS SMARTCARD_PPSExchange(uint8_t *ppsPtr)

1.5.1.2.8 SMARTCARD_ProtocolTypeGet Function

File

smart_card_layer3.h

Syntax

```
SMARTCARD_TRANSACTION_TYPES SMARTCARD_ProtocolTypeGet();
```

Description

This function gets the type of the protocol supported by the card.

Remarks

None

Preconditions

SMARTCARD_PowerOnATR was success

Function

SMARTCARD_TRANSACTION_TYPES SMARTCARD_ProtocolTypeGet(void)

1.5.1.2.9 SMARTCARD_Shutdown Function

File

smart_card.h

Syntax

```
void SMARTCARD_Shutdown();
```

Description

This function Performs the Power Down sequence of the SmartCard

Remarks

None

Preconditions

SMARTCARD_Initialize is called.

Example

```
statement1;
 if (resetRequest == WARM_RESET)
 {
     SMARTCARD_Shutdown();
 }
 ...
 ...
 ...

 // Not a Valid ATR Response
 scTransactionStatus = SC_ERR_BAR_OR_NO_ATR_RESPONSE;
 SMARTCARD_Shutdown();
 ...
 ...
 // Return False if there is no card inserted in the Slot or ATR of the card is unsuccessful
 if( !SCdrv_CardPresent() || (gCardState != ATR_ON ))
 {
     SMARTCARD_Shutdown();
     return SC_ERR_CARD_NOT_PRESENT;
 }
```

Function

void SMARTCARD_Shutdown(void)

1.5.1.2.10 SMARTCARD_StateGet Function

File

smart_card.h

Syntax

```
SMARTCARD_STATUS SMARTCARD_StateGet();
```

Description

This function returns the current state of SmartCard

Remarks

None

Preconditions

SMARTCARD_Initialize is called.

Example

```
main())
{
    SMARTCARD_Initialize();
    if(!SMARTCARD_StateGet());
    // Checks for Card ATR_ON state or Unknown state
    // If in ATR_ON state, then card can behave in normal manner, can communicate.
    // If in Unknown state the communication would time out and reset.
}
```

Return Values

Return Values	Description
SMARTCARD_UNKNOWN	No Card Detected
SMARTCARD_ATR_ON	Card is powered and ATR received

Function

SMARTCARD_STATUS SMARTCARD_StateGet(void)

1.5.2 Data types and constants

Enumerations

Name	Description
SMARTCARD_PPS_SUPPORT_STATUS	Protocol and Parameter Selections (PPS) status are defined
SMARTCARD_RESET_TYPES	Reset Types
SMARTCARD_STATUS	This Enum defines Answer To Reset(ATR) status
SMARTCARD_T0CASE_TYPES	Various cases handled under T=0 are defined
SMARTCARD_T1BLOCK_TYPES	This Enum defines the various blocks present in the T=1, Protocol Data Unit (PDU)
SMARTCARD_TRANSACTION_STATUS	Definition of various Error Types
SMARTCARD_TRANSACTION_TYPES	Transaction Protocol Types

Macros

Name	Description
EMV_SUPPORT	To Support the EMV standard part of the code
SMARTCARD_PROTO_T1	To enable the T1 protocol.
SMARTCARD_APDU_BUFF_SIZE	Define the Buffer size of Application Protocol Data Unit (APDU).
SMARTCARD_T1_PROTOCOL_MAX_BUFF_SIZE	Define the Maximum Buffer size for T1 Protocol.

Structures

Name	Description
SMARTCARD_APDU_COMMAND	This Structure defines APDU response packet
SMARTCARD_APDU_RESPONSE	This Structure defines response packet with status bytes
SMARTCARD_T1PROLOGUE_FIELD	This Structure defines Prologue field of T=1 protocol

1.5.2.1 SMARTCARD_APDU_COMMAND Structure

This Structure defines APDU response packet

File

smart_card.h

Syntax

```
typedef struct {
  uint8_t CLA;
  uint8_t INS;
  uint8_t P1;
  uint8_t P2;
  uint8_t LC;
  uint8_t LE;
} SMARTCARD_APDU_COMMAND;
```

Members

Members	Description
uint8_t CLA;	CLA Field :Command class
uint8_t INS;	INS Field :Instruction Operation code
uint8_t P1;	P1 Field : Selection Mode
uint8_t P2;	P2 Field : Selection Option
uint8_t LC;	LC Field : Data length
uint8_t LE;	LE Field : Expected length of data to be returned

Description

SmartCard APDU Response structure 7816-4

The structure defines the response packet definition of various data like CLA, INS, P1, P2,LC and LE

Remarks

None

1.5.2.2 SMARTCARD_APDU_RESPONSE Structure

This Structure defines response packet with status bytes

File

smart_card.h

Syntax

```
typedef struct {
  uint16_t rxDataLen;
  uint8_t apduData[512];
  uint8_t SW1;
  uint8_t SW2;
} SMARTCARD_APDU_RESPONSE;
```

Members

Members	Description
uint16_t rxDataLen;	Received Data length from smart card(excluding SW1 and SW2 bytes)
uint8_t apduData[512];	Application Protocol Data unit (APDU) max size in bytes is 512
uint8_t SW1;	Status byte 1

uint8_t SW2;	Status byte 2

Description

SmartCard APDU Response structure 7816-4

The APDU response byte which can hold 512 bytes of data and two status byte

Remarks

None

1.5.2.3 SMARTCARD_PPS_SUPPORT_STATUS Enumeration

Protocol and Parameter Selections (PPS) status are defined

File

smart_card.h

Syntax

```
typedef enum {
  SMARTCARD_PPS_NOT_ALLOWED,
  SMARTCARD_PPS_ALLOWED,
  SMARTCARD_PPS_ALLOWED_AFTER_WARM_RESET
} SMARTCARD_PPS_SUPPORT_STATUS;
```

Members

Members	Description
SMARTCARD_PPS_NOT_ALLOWED	Protocol Parameter Selection(PPS) not allowed (Specific Mode)
SMARTCARD_PPS_ALLOWED	Supports Parameter Parameter Selection (Negotiable Mode)
SMARTCARD_PPS_ALLOWED_AFTER_WARM_RESET	Supports Parameter Parameter Selection only after warm reset

Description

PPS support Status:

PPS support status states are defined

Remarks

provides various states for PPS support to track during protocol transaction

1.5.2.4 SMARTCARD_RESET_TYPES Enumeration

Reset Types

File

smart_card.h

Syntax

```
typedef enum {
  SMARTCARD_COLD_RESET,
  SMARTCARD_WARM_RESET
} SMARTCARD_RESET_TYPES;
```

Members

Members	Description
SMARTCARD_COLD_RESET	Cold Reset (Done only during Power ON)
SMARTCARD_WARM_RESET	Warm Reset

Description

Reset Type:

This Enum holds the type of Reset provided

Remarks

The enum type signifies the type of reset given by smart card operation

1.5.2.5 SMARTCARD_STATUS Enumeration

This Enum defines Answer To Reset(ATR) status

File

smart_card.h

Syntax

```
typedef enum {
  SMARTCARD_UNKNOWN,
  SMARTCARD_ATR_ON
} SMARTCARD_STATUS;
```

Members

Members	Description
SMARTCARD_UNKNOWN	Indicates the state before the reset of smartcard protocol
SMARTCARD_ATR_ON	Indicates the state after reset of smartcard protocol

Description

Smart card ATR Status:

It shows the whether ATR is ON or unknown byte

Remarks

None

1.5.2.6 SMARTCARD_T0CASE_TYPES Enumeration

Various cases handled under T=0 are defined

File

smart_card_layer3.h

Syntax

```
typedef enum {
  SMARTCARD_UNKNOWN_CASE,
  SMARTCARD_CASE_1,
  SMARTCARD_CASE_2S,
  SMARTCARD_CASE_2E1,
  SMARTCARD_CASE_2E2,
  SMARTCARD_CASE_3S,
  SMARTCARD_CASE_3E1,
  SMARTCARD_CASE_3E2,
  SMARTCARD_CASE_4S,
  SMARTCARD_CASE_4E1,
  SMARTCARD_CASE_4E2
} SMARTCARD_T0CASE_TYPES;
```

Members

Members	Description
SMARTCARD_UNKNOWN_CASE	An unknown mode under T=0
SMARTCARD_CASE_1	Case 1
SMARTCARD_CASE_2S	Case 2 Short Mode
SMARTCARD_CASE_2E1	Case 2 Extended Mode(1)
SMARTCARD_CASE_2E2	Case 2 Extended Mode(2)
SMARTCARD_CASE_3S	Case 3 Short Mode
SMARTCARD_CASE_3E1	Case 3 Extended Mode(1)
SMARTCARD_CASE_3E2	Case 3 Extended Mode(2)
SMARTCARD_CASE_4S	Case 4 Short Mode
SMARTCARD_CASE_4E1	Case 4 Extended Mode(1)
SMARTCARD_CASE_4E2	Case 4 Extended Mode(2)

Description

T0 Case types:

The various cases like short mode, extended mode are defined

Remarks

None

1.5.2.7 SMARTCARD_T1BLOCK_TYPES Enumeration

This Enum defines the various blocks present in the T=1, Protocol Data Unit (PDU)

File

smart_card_layer3.h

Syntax

```
typedef enum {
  SMARTCARD_NO_BLOCK,
  SMARTCARD_I_BLOCK,
  SMARTCARD_R_BLOCK,
  SMARTCARD_S_BLOCK,
  SMARTCARD_INVALID_BLOCK
} SMARTCARD_T1BLOCK_TYPES;
```

Members

Members	Description
SMARTCARD_I_BLOCK	I Block
SMARTCARD_R_BLOCK	R Block
SMARTCARD_S_BLOCK	S Block
SMARTCARD_INVALID_BLOCK	INVALID BLOCK

Description

Block Types in T=1:

In T=1 protocol, the PDU contains various blocks like I-block R-Block S-Block The above said blocks are defined.

Remarks

None

1.5.2.8 SMARTCARD_T1PROLOGUE_FIELD Structure

This Structure defines Prologue field of T=1 protocol

File

smart_card_layer3.h

Syntax

```
typedef struct {
  uint8_t NAD;
  uint8_t PCB;
  uint16_t length;
} SMARTCARD_T1PROLOGUE_FIELD;
```

Members

Members	Description
uint8_t NAD;	Node Address
uint8_t PCB;	Protocol Control Byte
uint16_t length;	LENGTH of I-Field

Description

Prologue Field for T=1 Protocol

None

Remarks

None

1.5.2.9 SMARTCARD_TRANSACTION_STATUS Enumeration

Definition of various Error Types

File

smart_card.h

Syntax

```
typedef enum {
  SMARTCARD_TRANSACTION_SUCCESSFUL = 1,
  SMARTCARD_ERROR_CARD_NOT_SUPPORTED = -16,
  SMARTCARD_ERROR_ATR_DATA,
  SMARTCARD_ERROR_NO_ATR_RESPONSE,
  SMARTCARD_ERROR_CMD_APDU_T0,
  SMARTCARD_ERROR_CMD_APDU_T1,
  SMARTCARD_ERROR_CARD_NOT_PRESENT,
  SMARTCARD_ERROR_CARD_NO_RESPONSE,
  SMARTCARD_ERROR_CARD_VPP,
  SMARTCARD_ERROR_PROCEDURE_BYTE,
  SMARTCARD_ERROR_PPS,
  SMARTCARD_ERROR_RECEIVE_CRC,
  SMARTCARD_ERROR_RECEIVE_LRC,
  SMARTCARD_ERROR_TRANSMIT,
  SMARTCARD_ERROR_T1_RETRY,
  SMARTCARD_ERROR_T1_S_BLOCK_RESPONSE,
  SMARTCARD_ERROR_T1_INVALID_BLOCK
} SMARTCARD_TRANSACTION_STATUS;
```

Members

Members	Description
SMARTCARD_TRANSACTION_SUCCESSFUL = 1	No Error

SMARTCARD_ERROR_CARD_NOT_SUPPORTED = -16	Card Not Supported
SMARTCARD_ERROR_ATR_DATA	ERROR in Answer-To-Reset (ATR) data received from the card
SMARTCARD_ERROR_NO_ATR_RESPONSE	No ATR Response from the card
SMARTCARD_ERROR_CMD_APDU_T0	Wrong T0 Command Application Protocol Data Unit (APDU)
SMARTCARD_ERROR_CMD_APDU_T1	Wrong T1 Command Application Protocol Data Unit (APDU)
SMARTCARD_ERROR_CARD_NOT_PRESENT	Card Not present in the slot
SMARTCARD_ERROR_CARD_NO_RESPONSE	No response from the card
SMARTCARD_ERROR_CARD_VPP	VPP Error received from the card (Voltage Mismatch/Programming Voltage not supported)
SMARTCARD_ERROR_PROCEDURE_BYTE	Incorrect Procedure Byte from the card
SMARTCARD_ERROR_PPS	Unsuccessful Protocol and Parameter Select (PPS) Exchange
SMARTCARD_ERROR_RECEIVE_CRC	CRC Error in the block received from the card
SMARTCARD_ERROR_RECEIVE_LRC	Longitudinal Redundancy check (LRC) Error in the block received from the card
SMARTCARD_ERROR_TRANSMIT	Transmission of byte to smart card failed
SMARTCARD_ERROR_T1_RETRY	Retry for T1 also Unsuccessful
SMARTCARD_ERROR_T1_S_BLOCK_RESPONSE	ERROR in T1 'S' Block Response
SMARTCARD_ERROR_T1_INVALID_BLOCK	Invalid block

Description

Smart Card Error Types:

During the protocol transaction various Errors could be encountered. This Enum defines the various errors states and it's interpretations.

Remarks

None

1.5.2.10 SMARTCARD_TRANSACTION_TYPES Enumeration

Transaction Protocol Types

File

smart_card_layer3.h

Syntax

```
typedef enum {
  SMARTCARD_T0_TYPE,
  SMARTCARD_T1_TYPE,
  SMARTCARD_INVALID_TYPE
} SMARTCARD_TRANSACTION_TYPES;
```

Members

Members	Description
SMARTCARD_T0_TYPE	T=0, Protocol is supported
SMARTCARD_T1_TYPE	T=1, Protocol is supported
SMARTCARD_INVALID_TYPE	Other than 0 or 1 its invalid

Description

Protocol Type Supported:

Two types of protocols supported

Remarks

The T=0/T=1 protocols are supported,this enum used in assigning the protocol type used in the smart card operation.

1.5.2.11 EMV_SUPPORT Macro

To Support the EMV standard part of the code

File

smart_card_config.h

Syntax

```
#define EMV_SUPPORT
```

Description

Support the EMV functionality in the code.

Remarks

None

1.5.2.12 SMARTCARD_PROTO_T1 Macro

To enable the T1 protocol.

File

smart_card_config.h

Syntax

```
#define SMARTCARD_PROTO_T1
```

Description

Support the T1 part of protocol in the code.

Remarks

None

1.5.2.13 SMARTCARD_APDU_BUFF_SIZE Macro

Define the Buffer size of Application Protocol Data Unit (APDU).

File

smart_card_config.h

Syntax

```
#define SMARTCARD_APDU_BUFF_SIZE 256
```

Description

Maximum size of the buffer of APDU

Remarks

None

1.5.2.14 SMARTCARD_T1_PROTOCOL_MAX_BUFF_SIZE Macro

Define the Maximum Buffer size for T1 Protocol.

File

smart_card_config.h

Syntax

```
#define SMARTCARD_T1_PROTOCOL_MAX_BUFF_SIZE 256
```

Description

Modify the T1 block buffer size as per the RAM size of the chosen micro & project requirement

Remarks

None

1.6 Demo

1.6.1 Configuring Hardware

This section describes how to set up the various configurations of hardware to run this demo.

1.6.1.1 Configuration using Explorer 16 Board

Following are the Hardware Required :

1. Explorer 16 (Microchip part number DM240001)
2. SC (Smart/Sim Card) PICTail Board
3. And one of the following PIMs

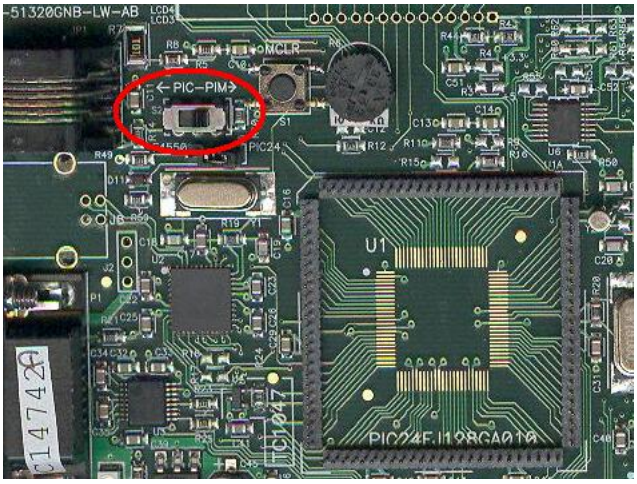
1. PIC24FJ256GB110 Plug-In-Module (PIM) (Microchip part number MA240014),

2. PIC24FJ128GB204 Plug-In-Module (PIM) (Microchip part number MA240036),

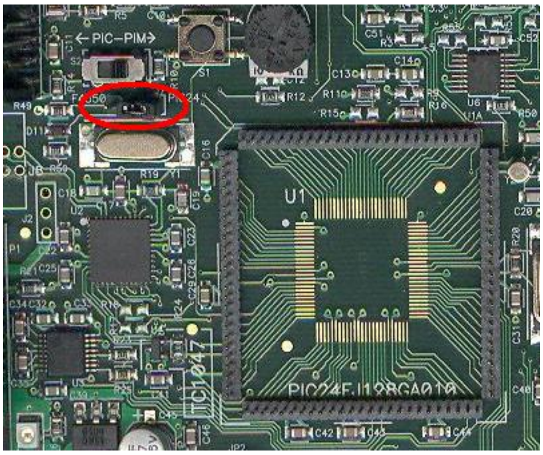
3. PIC24FJ128GA204 Plug-In-Module (PIM) (Microchip part number MA240037),

Steps

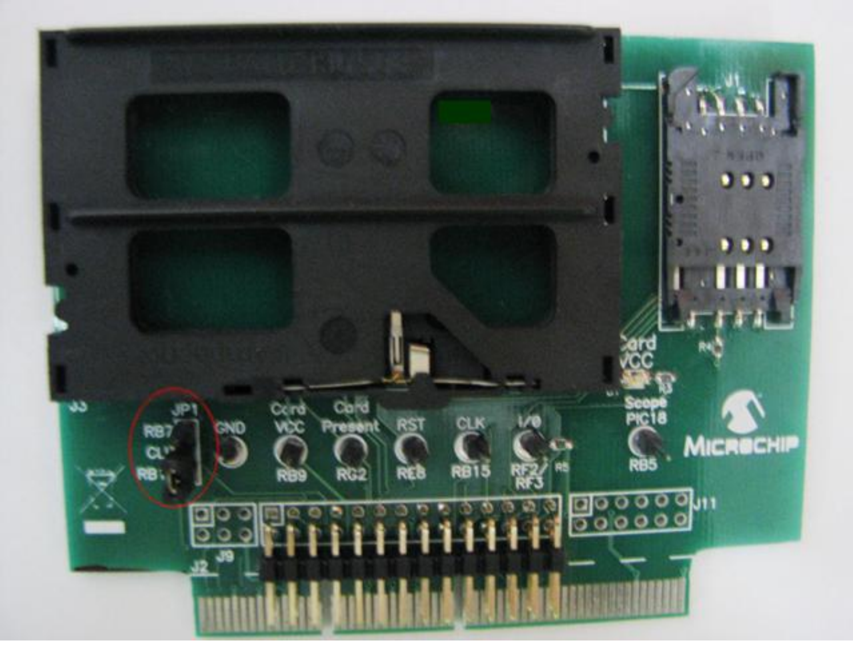
1. Before attaching the PIM to the Explorer 16 board, insure that the processor selector switch (S2) is in the “PIM” position as seen in the image below.



2. Short the J7 jumper to the “PIC24” setting



3. Be careful while inserting the PIC24FJ256GB110 PIM or any other appropriate PIM into Exp 16 board. Insure that no pins are bent or damaged during the process. Also insure that the PIM is not shifted in any direction and that all of the headers are properly aligned.
4. Short JP1 to SRC1 (i.e. RD1) or SRC2 (i.e. RB15) based upon the smart card clock pin configured in the firmware: Example: - Short JP1 to SRC1 while using PIC24FJ256GB110 demo.



5. Insert the J2 slot of SC (Smart/Sim Card) PICTail card into J5 port of Explorer 16 board. Make sure that the Smart Card Connector is facing towards the Explorer 16 board. Insert the Smart Card in SC PICTail board.

1.6.1.2 Resource Usage - PIC24F

These tables specify the program memory, execution speed, RAM usage, and build requirements PIC24F devices.

Program Memory

Module	Optimization	Program Memory
Smart Card Library (T=0)	-O0	2.7K
Smart Card Library (T=0)	-O1	1.9K
Smart Card Library (T=0)	-Os	1.7K
Smart Card Library (T=0 and T=1)	-O0	4.7K

Smart Card Library (T=0 and T=1)	-O1	3.2K
Smart Card Library (T=0 and T=1)	-Os	2.9K

RAM Usage (bytes)

Module/Layer	Global	Stack	Heap
Smart Card Library (T=0)	300	Not available	None
Smart Card Library (T=0 and T=1)	330	Not available	None

Peripherals

Type/Use	Specific/Configurable	Polled/Interrupt	Limitations
UART	Select via Programming, Tx and Rx Signals	Polled	None
Card Power Output	Select via drv_smart_card_sw.h	Polled	Be able to source sufficient current to power the Smart Card
Card Reset Output	Select via drv_smart_card_sw.h	Polled	Totempole or Open Drain with pullup
Card Present Input	Select via drv_smart_card_sw.h	Polled	Input with Pullup
Clock Output	REFO Output	n/a	Clock Output to Card should be close to 4MHz (3.57MHz for exact Baud Rate, but not required)

Build Requirements

None

1.6.2 Run Demo

1.6.2.1 Getting Started - Smart Card Demo

This demo shows how the smart card library for PIC microcontroller is used to communicate a smart card using T = 0 and T = 1 protocols. The demo has to be run in the debug mode of MPLAB IDE.

1.6.2.2 Firmware

To run this project, you will need to load the corresponding firmware into the devices.

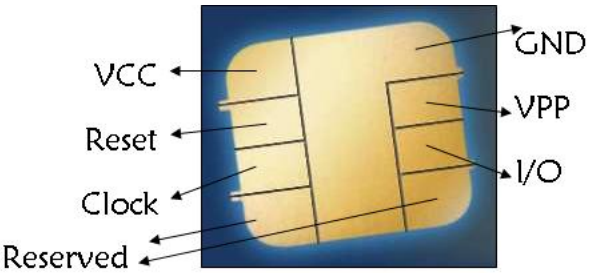
The source code for this demo is available in the “<Microchip Solutions\apps\smartcard\firmware\src\smart_card_demo_main.c” directory. In this directory you will find all of the user level source and header files, linker file as well as project file for each of the hardware platforms. Find the project (*.mcp) file that corresponds to the hardware platform you wish to test. Compile and program the demo code into the hardware platform. For more help on how to compile and program projects, please refer to the MPLAB® IDE help available through the help menu

of MPLAB (Help->Topics…->MPLAB IDE).

1.6.2.3 Running the Demo

This demo uses the selected hardware platform as a Smart card reader. The demo has to be run in the debug mode of MPLAB IDE. Please refer "Configuring the Hardware" section for the bench setup connections.

Smart Card consists of 8 pins namely:-



I/O: Input or Output for serial data to the integrated circuit inside the card.

VPP: Programming voltage input (optional use by the card).

GND: Ground (reference voltage).

CLK: Clocking or timing signal.

RST: Reset Signal to the Card.

VCC: Power supply input (optional use by the card).

Communication between the interfacing device and smart card is done as per the following steps:-

1. Insertion of the smart card in the slot.
2. Detection of the smart card insertion by the microcontroller (interfacing device).
3. Microcontroller does the cold reset of the smart card.
4. Answer to Reset (ATR) response by the card.
5. PPS exchange (if smart card supports it).
6. Execution of the transaction(s) between the card and the interfacing device.
7. Removal of the smart card from the slot.
8. Detection of the smart card removal by the microcontroller.
9. Deactivation of the contacts.

Contact type smart card communication protocols that are generally used are:-

- T = 0 asynchronous half duplex character transmission.
- T = 1 asynchronous half duplex block transmission.

The data transfers between the card and the terminal happens on the single wire I/O line. The smart card library supports both T=0 and T=1 protocol.

Example code for T=0 cards:-

The demo executes the card command Selecting Payment System Environment(PSE) master file "1PAY.SYS.DDF01"

and displays the card record details on UART terminal.

If the master file is not found then the demo application will try to select the application using application identifier(AID). Currently demo supports the AID query for Master Card , VISA , Amex and Maestro

Example code for T=1 cards:-

The demo executes the "Get CPLC (Card Production Life Cycle) data" command for T=1 java card. The command list can be extended further as per the smart card manual and the project requirement.

The demo waits in the while(1) loop until the smart card is inserted in the smart card connector slot. Once the card is inserted in the slot, 'Cold Reset' and 'PPS' (Protocol and Parameter Selection) has to be performed to the smart card running MPLAB project in debug mode. If the user has inserted T=0 card in the slot, then “SMARTCARD_EMV_DataExchangeT0” function is called and the result of the executed command from the smart card is stored in “apduData”. If the user has inserted T=1 card in the slot, then “SMARTCARD_EMV_DataExchangeT1” function is called and the result of the executed command from the smart card is stored in “apduData”.

Variable “cardResponse” stores the status codes and the length of the received data from the smart card.

Note: After initially being reset by the card reader, the smart card responds with a string of characters known as the Answer to Reset, or ATR. These characters consist of an initial character, TS, followed by a maximum of 32 additional characters. Together, these characters provide information to the card reader about how to communicate with the card for the remainder of the session. If the card reader wants to modify the data transmission parameters in the smart card, then it must perform PPS in accordance with Smart Card EMV standards before the transmission protocol is actually used.

For more details about smart card communication using PIC microcontrollers, please refer the application note AN1370

1.6.2.4 Configuring the pins

The current Smart Card software library supports 16-bit PIC microcontrollers.The port pins connection between the micro and smart card is defined in "smart_card_config" file. The demo uses the signal connections between the smart card and PIC microcontroller port pins as per the below table:-

Signal Name	PIC24FJ256GB110	PIC24FJ128GB204/GA204
SIM_CARD_DET	RB1	RA1
SMART_CLK	RC7	RC7
SMART_I/O	RC4,RF2	RB7,RB13
SMART_RST	RE8	RC2
SMART_CARD_DET	RB0	RA0
SMART_VCC	RB9	RB15

“SMART_CARD_DET”/”SIM_CARD_DET” signals indicate the presence of Smart Card/Sim Card to the microcontroller. Either of one between Smart Card and Sim Card has to be inserted in the Smart Card PICTail board. If both the cards are inserted at a time in the PICTail card, then the demo won’t work successfully.

If the user wants to connect the smart card signals to different port pins of the micro, then the pin mapping in “smart_card_pps_macro.h” file needs to be modified.

Index