FS2711IC-PD-PPS Application Notes

V1.0 2020.10.6

Hardware

The chip can be powered directly by VDD connect to VBUS, the pin is up to 31V tolerated. V3P3 can output up to 50mA current to power other chip of the system, the pin is required to connect a capacitor, value like 0.47uA or 1uA.
When system connect to PD adapter, the PD adapter can power the system through VBUS.

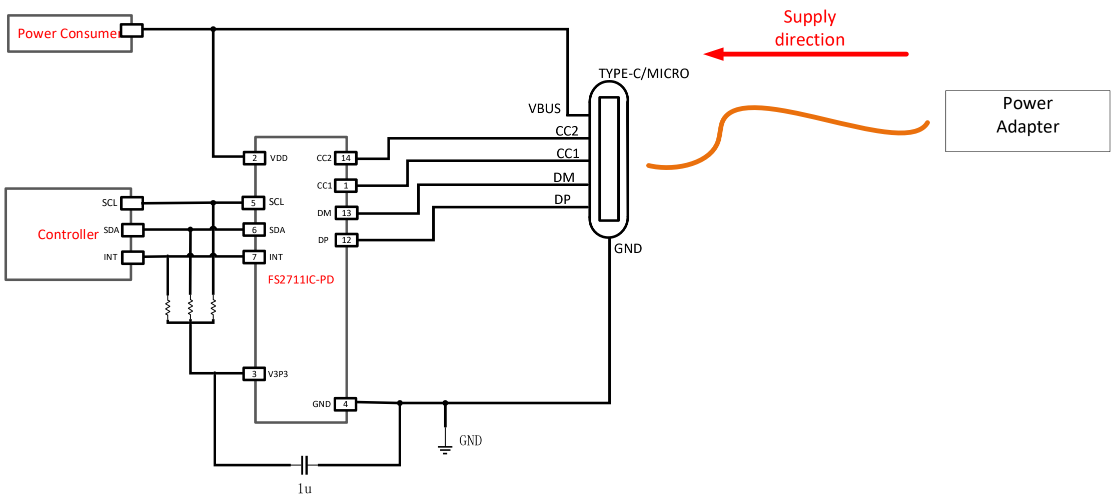
I2C

The address of FS2711IC is 0x5A(8 bit address.) or 0x7d(7 bit address, add read bit is 0x5B, write bit is 0x5A).
Support random address R/W and continuous R/W up to 16Byte.
I2C can support up to 3.3MHz speed, the recommended is 400K.
Three pins are available, SCL/SDA/INT for SOP16, two pins SCL/SDA for SSOP10 package. Those pins should connect with resistor to V3P3 pin.

I2C speed	***Rup***
100KHz	10KΩ
400KHz	4.7KΩ （建议）
1MHz	1KΩ

For INT pin, the valid value is from high to low, user can call int process function to get interrupt information.

Application diagram



Software

FASTSOC has C based lib for user to fast their development with FS2711IC. One is fssk_lib.h and another is fssk_lib.c, which are list in this document. They encapsulate most lower level hardware operation.

FSSK_LIB_t

The structure defined common used operation variable.

```
//FSSK Struct
typedef struct{
	//PD PDO
	u8	pdo_num;
	u8	pdo_type;
	u8	req_pdo_num;
	u16 pdo_max_volt[7];
	u16 pdo_min_volt[7];
	u16 pdo_max_curr[7];
}FSSK_LIB_t;
```

pdo_num： used to store number of PD PDO, from 1 to 7.
pdo_type： 7bit to indicate type of each PD PDO, 1 for PPS type, 0 for FIXED type.
pdo_max_volt/ pdo_min_volt/ pdo_max_curr： each used to store start voltage, end voltage and max current for each PD PDO. If it is fixed PDO, start voltage is equal to end voltage. The

unit of voltage is mV, the unit of current is mA.

Reset function

FSSK_System_Reset：reset the whole chip.
FSSK_Port_Reset: reset the typec protocol, Rd of CC will disconnect and re-connect again.

Protocol selection function

FSSK_Select_Protocol: select one supported protocol.
FSSK_Enable_Protocol: Enter or quit the selected protocol.

The supported protocol include: PD, PPS, QC2/3 CLASSA/B.

Voltage/Current selection function

FSSK_Select_Voltage_Current: set the voltage and current of selected protocol.
Define fssk_lib.req_pdo_num before the function calling to select the PDO.

start_volt/end_voltage

For PD：no need to config.
For PPS: should config voltage.
For QC: the start voltage/end voltage, unit is mV.

To select a fix voltage, the start_volt = end_volt. For example, start_volt=end_volt=9000, and select QC protocol, the chip will apply QC 9V. If end_volt > start_volt, end_volt = 12000, start_volt = 5000 and select QC protocol, the chip will apply QC and sweep 5V-9V-12V-5V-9V-12V with fixed period.

max_curr

Set current for selected protocol. Unit is mA，only valid for PD/PPS.

Example1 – Fixed PDO

FSSK_Select_Protocol(FSSK_PD);
FSSK_Enable_Protocol(1);

The Code will select PD protocol, the interrupt function will be called by INT pin trigger. User can get adapter PD information from FSSK_LIB_t structure variable.
Assume adapter PDO is 5V3A 9V2A 12V1.5A.

fssk_lib.req_pdo_num = 1;
FSSK_Select_Voltage_Current(FSSK_PD, 0, 0, 2000); //apply 9V2A
FSSK_Enable_Voltage_Current();

The code will select PD protocol and select PDO2, with current 2A, voltage maybe 9V depend on the adapter broadcast(those information and values are read from interrupt function).

fssk_lib.req_pdo_num = 2;
FSSK_Select_Voltage_Current(FSSK_PD, 0, 0, 1500); //apply 12V1.5A
FSSK_Enable_Voltage_Current();

Example2 – PPS PDO

FSSK_Select_Protocol(FSSK_PD);
FSSK_Enable_Protocol(1);

The code first select FSSK_PD, although adapter has PPS PDO. User will get adapter information from interrupt service function.

Assume adapter PDO is: 5V3A 9V2A 12V1.5A 3.3~5.9V2A 3.3~12V1.5A

FSSK_Select_Protocol(FSSK_PPS);
fssk_lib.req_pdo_num = 3; // select 3.3~5.9V2A
FSSK_Select_Voltage_Current(FSSK_PPS， 4500, 4500, 1500); //apply 4.5V1.5A
FSSK_Enable_Protocol(1);

2nd select and apply code:
fssk_lib.req_pdo_num = 3; // select 3.3~5.9V2A
FSSK_Select_Voltage_Current(FSSK_PPS， 5500, 5500, 2000); //apply 5.5V2A
FSSK_Enable_Voltage_Current();

3rd select and apply code:

```
fssk_lib.req_pdo_num = 4; // select 3.3~12V1.5A
FSSK_Select_Voltage_Current(FSSK_PPS,  11000, 11000, 1500); //apply 11V1.5A
FSSK_Enable_Voltage_Current();
```

fssk_lib.h

```
#ifndef FSSK_LIB_H
#define FSSK_LIB_H

//For 8051 CPU
#define u8   unsigned char
#define u16 unsigned int
#define u32 unsigned long

//Protocol type
#define FSSK_PPS                20
#define FSSK_PD                 21
#define FSSK_QC2A                4
#define FSSK_QC2B                5
#define FSSK_QC3A                6
#define FSSK_QC3B                7

//PDO type
#define FSSK_PDO_FIX            0
#define FSSK_PDO_PPS            1

//Capability type
#define FSSK_MAX_5V             1
#define FSSK_MAX_9V             3
#define FSSK_MAX_12V            7
#define FSSK_MAX_20V            15

//Sweep mode
#define FSSK_SWEEP_SAW          0
#define FSSK_SWEEP_TRI          1
#define FSSK_SWEEP_STEP         2

//FSSK Struct
typedef struct{
        //PD PDO
        u8   pdo_num;
        u8   pdo_type;
```

```
	u8    req_pdo_num;
	u16 pdo_max_volt[7];
	u16 pdo_min_volt[7];
	u16 pdo_max_curr[7];
}FSSK_LIB_t;

void FSSK_Debug();
void FSSK_System_Reset();
void FSSK_Port_Reset();
void FSSK_Select_Protocol(u8 protocol);
void FSSK_Enable_Protocol(u8 ena);
void FSSK_Select_Voltage_Current(u8 protocol, u16 start_volt, u16 end_volt, u16 max_curr);
void FSSK_Enable_Voltage_Current();
void FSSK_Set_Sweep_Mode(u8 mode);
void FSSK_IRQHandler();

#endif
```

fssk_lib.c

```
#include "fssk_lib.h"

FSSK_LIB_t fssk_lib;
/*
Check if Parameter is Valid
*/
void FSSK_Debug(){
	while(1);
}
/*
Reset FSSK
*/
void FSSK_System_Reset(){
	Write_IIC(0x4a, 0x1);
}
/*
Reset C Port
*/
void FSSK_Port_Reset(){
	Write_IIC(0xa0, 0x2);
	Write_IIC(0x49 ,0x0);
	delay_ms(1);
```

```
        Write_IIC(0xa0, 0x0);
        Write_IIC(0x49 ,0x1);
}
/*
Enter or Exit a Protocol
*/
void FSSK_Select_Protocol(u8 protocol){
        if(protocol>21){
                FSSK_Debug();
        }
        Write_IIC(0x42, protocol);
}

void FSSK_Enable_Protocol(u8 ena){
        if(ena){
                Write_IIC(0x41, 0x1);
        }else{
                Write_IIC(0x41, 0x2);
        }
}
/*
Request Voltage & Current of a protocol
*/
void FSSK_Select_Voltage_Current(u8 protocol, u16 start_volt, u16 end_volt, u16 max_curr){
        u16 wr_l, wr_h;
        if(protocol>21){
                FSSK_Debug();
        }
        if(start_volt > end_volt){
                FSSK_Debug();
        }
        switch(protocol){
                case FSSK_QC2A:
                case FSSK_QC2B: {
                                if(start_volt == 5000){
                                        wr_l = 0x0;
                                }else if(start_volt == 9000){
                                        wr_l = 0x1;
                                }else if(start_volt == 12000){
                                        wr_l = 0x2;
                                }else if(start_volt == 20000){
                                        wr_l = 0x3;
                                }else{
                                        FSSK_Debug();
                                }
                                if(end_volt == 5000){
                                        wr_h = 0x0;
                                }else if(end_volt == 9000){
                                        wr_h = 0x1;
                                }else if(end_volt == 12000){
                                        wr_h = 0x2;
                                }else if(end_volt == 20000){
                                        wr_h = 0x3;
                                }else{
                                        FSSK_Debug();
                                }
                                break;
                            }
                            case FSSK_QC3A:
                            case FSSK_QC3B: {
                                wr_l = (start_volt - 3600)/200;
                                wr_h = (end_volt - 3600)/200;
                                break;
                            }
                case FSSK_PD:
                case FSSK_PPS: {
                            Write_IIC(0x46, fssk_lib.req_pdo_num + (fssk_lib.req_pdo_num<<4));
                            wr_l = (start_volt - fssk_lib.pdo_min_volt[fssk_lib.req_pdo_num])/20;
                            wr_h = (end_volt - fssk_lib.pdo_min_volt[fssk_lib.req_pdo_num])/20;
                            Write_IIC(0xde, max_curr/50);
                            break;
                }
                default: FSSK_Debug();
        }
        Write_IIC(0xf4, wr_l&0xff);
        Write_IIC(0xf5, (wr_l>>8)&0xff);
        Write_IIC(0xf6, wr_h&0xff);
        Write_IIC(0xf7, (wr_h>>8)&0xff);
}

void FSSK_Enable_Voltage_Current(){
        Write_IIC(0x43, 0x1);
}

/*
Set Sweep Mode
*/
void FSSK_Set_Sweep_Mode(u8 mode){
	Write_IIC(0x47, mode);
}

/*
FSSK IRQ Handler
*/
void FSSK_IRQHandler(){
	u8 i2c_rdata;
	u8 pdo_b0, pdo_b1, pdo_b2, pdo_b3;
	u8 i;
	i2c_rdata = ~Read_IIC(0xb1); //need read at least once
	i2c_rdata = ~Read_IIC(0xb2);

	if(i2c_rdata & 0x2){ //PDO Update
		fssk_lib.pdo_num = 0;
		fssk_lib.pdo_type = 0;
		for(i=0;i<7;i++){
			pdo_b0 = Read_IIC(0xc0+i*4);
			pdo_b1 = Read_IIC(0xc1+i*4);
			pdo_b2 = Read_IIC(0xc2+i*4);
			pdo_b3 = Read_IIC(0xc3+i*4);
			if(pdo_b0){
				if(pdo_b3 & 0xc0){
					fssk_lib.pdo_type |= 1<<i;
					fssk_lib.pdo_min_volt[i] = pdo_b1*100;
					fssk_lib.pdo_max_volt[i] = ((pdo_b2>>1) + ((pdo_b3&0x1)<<7))*100;
					fssk_lib.pdo_max_curr[i] = (pdo_b0 & 0x7f)*50;
				}else{
					fssk_lib.pdo_min_volt[i] = ((pdo_b1>>2) + ((pdo_b2&0xf)<<6))*50;
					fssk_lib.pdo_max_volt[i] = fssk_lib.pdo_min_volt[i];
					fssk_lib.pdo_max_curr[i] = (pdo_b0 + ((pdo_b1&0x3)<<8))*10;
				}
			}else{
				break;
			}
		}
		fssk_lib.pdo_num = i;
	}
}
```