i2cTalkToLDC1612.c README

Authors: Chen, Hongkun and Chen, Yiming and Chen, Jiahao on July 2024 @ m&m lab

Introduction

To use a I2C facility, like LDC162, to talk to TI's 28377xD, **it is needed to configure the register properly of 28377xD in accurate time.** We need to configure five main parts of functions, which are

initial fucntion

```
void I2CA_Init(void)
```

read I2C data function

```
int I2cRead16bitData(Uint16 SlaveRegAddr)
```

write I2C data function

```
int I2cWrite16bitData(Uint16 ConfigRegAddr, Uint16 value)
```

channel configuration function

```
int Single_channel_config(Uint16 channel)
```

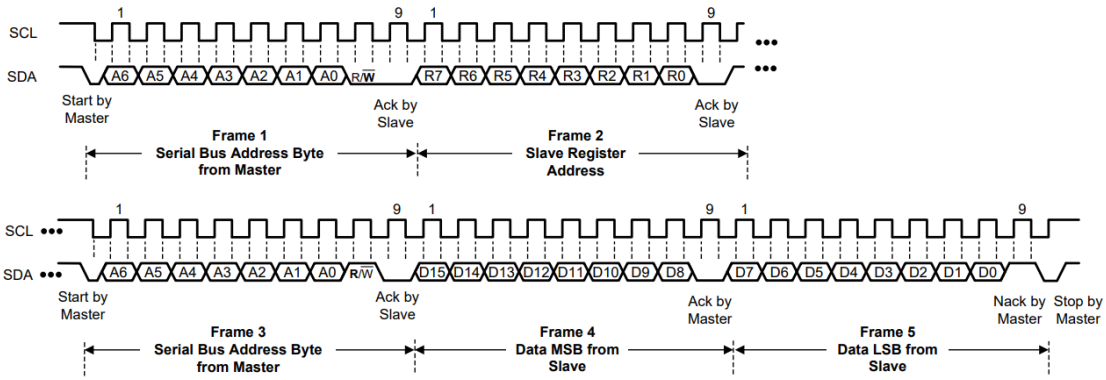
read channel function

```
Uint32 I2CA_ReadData_Channel(Uint16 channel)
```

In this README, we are about to start from the basic concept of I2C, then we are going to unscramble the code.

Basic concept of I2C

I2C read processing



Pay attention: The following configuration reference is TMS320F2837xD Dual-Core Delfino Microcontrollers Technical Reference Manual, the paragraph that needs to be cited indicates only the page on which the document is cited. If you are acquainted with I2C communication and configuration, just skip it!

We start introducing the I2C read processing, like the figure above. This processing are divided into five frame. According to 28377xD's manual script, we can finish I2C communication by setting the inherent registers in 28377xD in the right timing. The registers we might use are listed below.

- I2CSAR: I2C Slave Address
- I2CPSC: I2C Presclaer
- I2CCLKL: I2C Clock low-time divider
- I2CCLKH: I2C Clock high-time divider
- I2CIER: I2C Interrupt Enable
- I2CMDR: I2C Mode Setting
- I2CFFTX: I2C FIFO Transmit Register
- I2CFFRX: I2C FIFO Receive Register
- I2CSTR: I2C Status
- I2CCNT: I2C Data Count
- I2CDXR: I2C Data Trasmit

Before the frame 1 begining, it is naturally to configure the address of slave facility, I2C system clock, interrupt requests, the working mode of I2C facility (Master or Slave?), the FIFO register. So, we can carry out the above work in initial function.

We can let I2CSAR.all equals to the address of slave facility, as for LDC1612 who's ADDR pin connected to high voltage (3.3V), the slave facility is 0x2B.

```
I2caRegs.I2CSAR.all = I2C_SLAVE_ADDR;
```

As for an instance, we set I2CSPC equals 8 MHz. According to formula: Frequency = SYSCLKOUT / (I2CCLKL+I2CCLKH), if the target I2C frequency is 100kHz, to make the high and low levels is 50:50, we can let I2CCLKL and I2CCLKH both equal to 40.

```
I2caRegs.I2CPSC.all = 8;
I2caRegs.I2CCLKL = 40;
I2caRegs.I2CCLKH = 40;
```

Then to enable stop condition detected and register access ready, we must configure the I2CIER, iwe let bit 2 and bit 5 both be 1.

```
I2caRegs.I2CIER.all = 0x24;
```

Then, as for 28377xD is actually a mater before starting to communication, whether we choose to write or read data through I2C, and it must transmit a frame which contains the address of slave's address, finally we want to enable the setting of I2C mode setting. So:

```
I2caRegs.I2CMDR.bit.MST = 1;
I2caRegs.I2CMDR.bit.TRX = 0;
I2caRegs.I2CMDR.bit.IRS = 1;
```

Then, we can configure the FIFO registers to enable to work:

```
I2caRegs.I2CFFTX.all = 0x6000;
I2caRegs.I2CFFRX.all = 0x2040;
```

First, we are going to check the I2C bus is in stop condition and is busy. If we miss this procedure, we might face the communication conflict condition.

```
if (I2caRegs.I2CMDR.bit.STP == 1)
    {

       return I2C_STP_NOT_READY_ERROR;
    }
//...
if (I2caRegs.I2CSTR.bit.BB == 1)
    {

         return I2C_BUS_BUSY_ERROR;
    }
```

In frame 1, I2C transmit serial bus address byte from master.

```
 I2caRegs.I2CSAR.all = 0x2B;
```

What we want to emphasis is the difference between data and address properties in I2C.

In the above figure, it is stipulated that only serial bus address byte from master has the real address properties. Slave register address, and data MSB/LSB share the data properties. Before transmitting the frame data, it is accustomed to initialize the register I2CCNT to tell the 28377xD how many frame it will transmit before the next serial bus address byte from master transmitting.

So, on the basis of above exposition, we can know in frame 2:

```
I2caRegs.I2CCNT = 1;
I2caRegs.I2CDXR.all = SlaveRegAddr;
```

And, we need to tell the mode of I2C bus:

```
I2caRegs.I2CMDR.all = 0x2620;
```

It means, in frame 2, I2C module is enabled, and it is in master & transmitter mode (it generates the serial clock on the SCL pin and transmits data on the SDA pin), meanwhile, it starts to transmit data, but in this frame I2C does not have any stop condition, so bit 11 is actually 0.

Then, in frame 3, just like frame 1 and frame 2

```
I2caRegs.I2CSAR.all = 0x2B;
I2caRegs.I2CCNT = 2;
I2caRegs.I2CMDR.all = 0x2C20;
```

I2CCNT is 2 means before the next serial bus address byte from master transmitting we want to transmit 2 frame data (frame 4 and frame 5, Data MSB from Slave and Data LSB from Slave).

Making I2CMDR equal to 0x2C20 signifies that 28377xD is no longer a transmitter, but it still a master. After frame 5, this master have to send a stop condition bit, which is bit 11.

As the data received from LDC1612 is 32 bit, this 32 bit contains 2 parts: 16 bit MSB and 16bit LSB, the slave register address corresponding to MSB and LSB is diverse. Take an instance, when we just use the channel 0, that is the resource channel of DATA 0, the slave register address corresponding to MSB of this channel is 0x00, the slave register address corresponding to LSB of this channel is 0x01. So, when 28377xD reading channel 0 data, it has to invoke below function twice, and deliver 0x00 as SlaveRegAddr first then deliver 0x01 as SlaveRegAddr, as LDC1612 have stipulate we must read MSB first.

```
int I2cRead16bitData(Uint16 SlaveRegAddr)
```

For 16 bit MSB/LSB, we need to use buffer to complete bit shift and logical operations. The condition in while loop means when receive FIFO is empty, we start to process the MSB/LSB.

```
while(I2caRegs.I2CFFRX.bit.RXFFST == 0);
DataBuffer = I2caRegs.I2CDRR.all;
DataBuffer = DataBuffer << 8;
while(I2caRegs.I2CFFRX.bit.RXFFST == 0);
DataBuffer |= I2caRegs.I2CDRR.all;
```

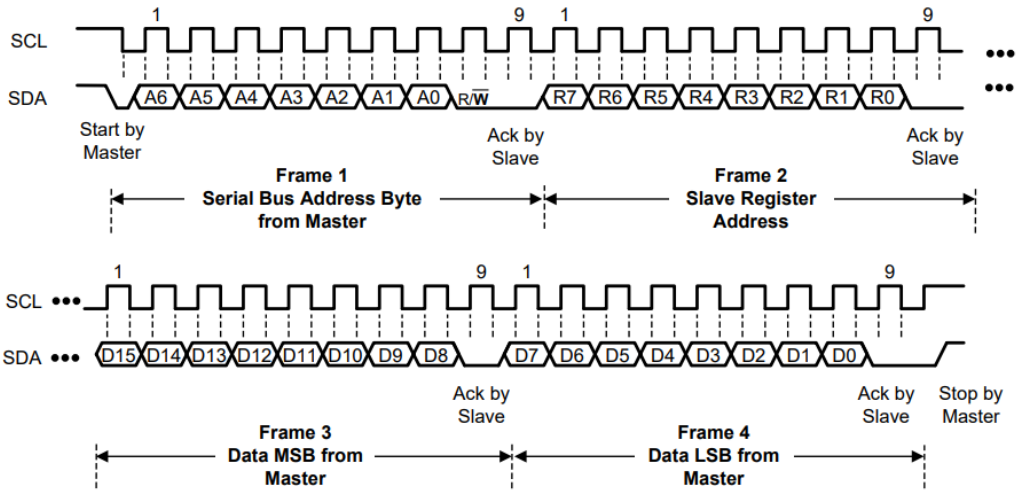
Things you have to pay attention

After configuring above, you have to continue writing this:

```
DELAY_US(50);
```

The meaning of this code is to give the I2C bus idle time, ensure that all previous frames have been read, and there will be no overlap in the data demodulation process resulting in garbled characters.

I2C write processing



most configuration is just like I2C read processing, we just list the discrepancy here.

```
I2caRegs.I2CCNT = 3;
```

This means, in writing processing at a time, we just use serial bus address byte from master once. So, frame 2, frame 3 and frame 4 are successive data property frame, just set I2CCNT = 3.

```
I2caRegs.I2CMDR.all = 0x6E20;
```

This means, as the figure shows, when in 12C write processing, I2C module of 28377xD is enabled and it is a master & transmitter, in this frame 2, frame 3 and frame 4 successive data property frame, I2C has to accomplish start operation, stop operation.

We no longer need the FIFO service in the process of write processing, we use I2CSTR instead.

```
data[1] = value & 0x00ff;
data[0] = value >> 8;
while(I2caRegs.I2CSTR.bit.XRDY == 0);
I2caRegs.I2CDXR.all = data[0];
while(I2caRegs.I2CSTR.bit.XRDY == 0);
I2caRegs.I2CDXR.all = data[1];
```

Things you have to pay attention

After configuring above, you have to continue writing this:

```
DELAY_US(50);
```

The meaning of this code is to give the I2C bus idle time, ensure that all previous frames have been read, and there will be no overlap in the data demodulation process resulting in garbled characters.

The return value of this part of the function should also not be 1, but 0, to fit the subsequent LDC1612 initialization of the function.

```
return 0;
```

LDC1612 initial configuration

This part is mainly derived from Seeed's studio. We only put some part you may interested.

```
Set_Rp(CHANNEL_0, 15.727);
Set_L(CHANNEL_0, 18.147);
Set_C(CHANNEL_0, 100);
Set_Q_factor(CHANNEL_0, 35.97);
```

In this part, you will need to modify the resistance, inductance, capacitance, and quality factor according to the data obtained by Texas Instruments Coil Designer.

Initial configuration is required for each LDC1612 channel.