CAN PROTOCOL IMPLEMENTATION (LOW LEVEL CAN DRIVER/FIRMWARE) USING LPC2129 uC & Keil Embedded C for ARM LEVEL 1

Pre requisites

- Knowledge of Embedded C Programming for ARM7 arch using Keil IDE for LPC2129 uC.
- Exposure to Startup.s auto-generated file
- Exposure to PLL,VPB Divider,MAM ,Pin Connect Block Configurations
- GPIO Operations

Quick Walk Through ARM Arch

- The arcronym “ARM” stands for Advanced RISC Machine.
- It refers to a family of processor cores that was introduced in the early 80’s by Acorn Computers Ltd,later renamed as ARM Ltd.
- Has been constantly enhanced through time to meet demanding needs of application in the embedded market.
- It accounts for more than 80% the embedded market.
- ARM Cores are ARM 1 to ARM 11,ARM Cortex A,R,M Series.
- ARM 1 to ARM 7 – Von Neuman Architecture,
- ARM 8 Onwards – Harvard Architecture
- Are Available as 32bit,64bit cores
- Are Available as single,dual,quad,octa ….Cores
- Popularity of ARM Arch began with ARM 7 Core used for the Dawn of the Cell/Mobile Phone Era

Quick Walkthrough ARM7 Arch

- ARM7 is a 32-bit single core processor
  - Von-Neuman,RISC
  - 32 bit ALU,
  - 32-bit Address Bus,
  - 32-bit Data Bus
  - 37,32 bit Core Registers
  - 7 Modes of Operations
  - 3- Pipeline Stages
  - 6 Sources of Interrupts for Core
  - JTAG,ETM,RTM – Debugging
  - 2-Sets of Intructions ARM-ISA & THUMB-ISA

Quick Walkthrough ARM7 Arch

- ❖ Supports Conditional Executions of Instructions
- ❖ Support a Separate Fast Multiplier With Accumulating Unit
- ❖ Support a Separate Incrementer Unit
- ❖ Support an Inline Barrel Shifter Unit
- ❖ Presence of the above 3 units leads to Instruction Fusion.

LPC2129 (ARM-7 Core)uC : Feature Set

1)ARM7TDMI-S based high-performance single core,32-bit RISC Microcontroller with Thumb extensions

2)256KB on-chip Flash ROM with ISP & IAP

256*1024= 262144bytes/40000h bytes

(0x00000000 to 0x0003ffff)

3)16KB SRAM,

16 * 1024= 16384bytes/4000h bytes

(0x40000000 to 0x40003fff)

5)General purpose I/O pins (upto 46).

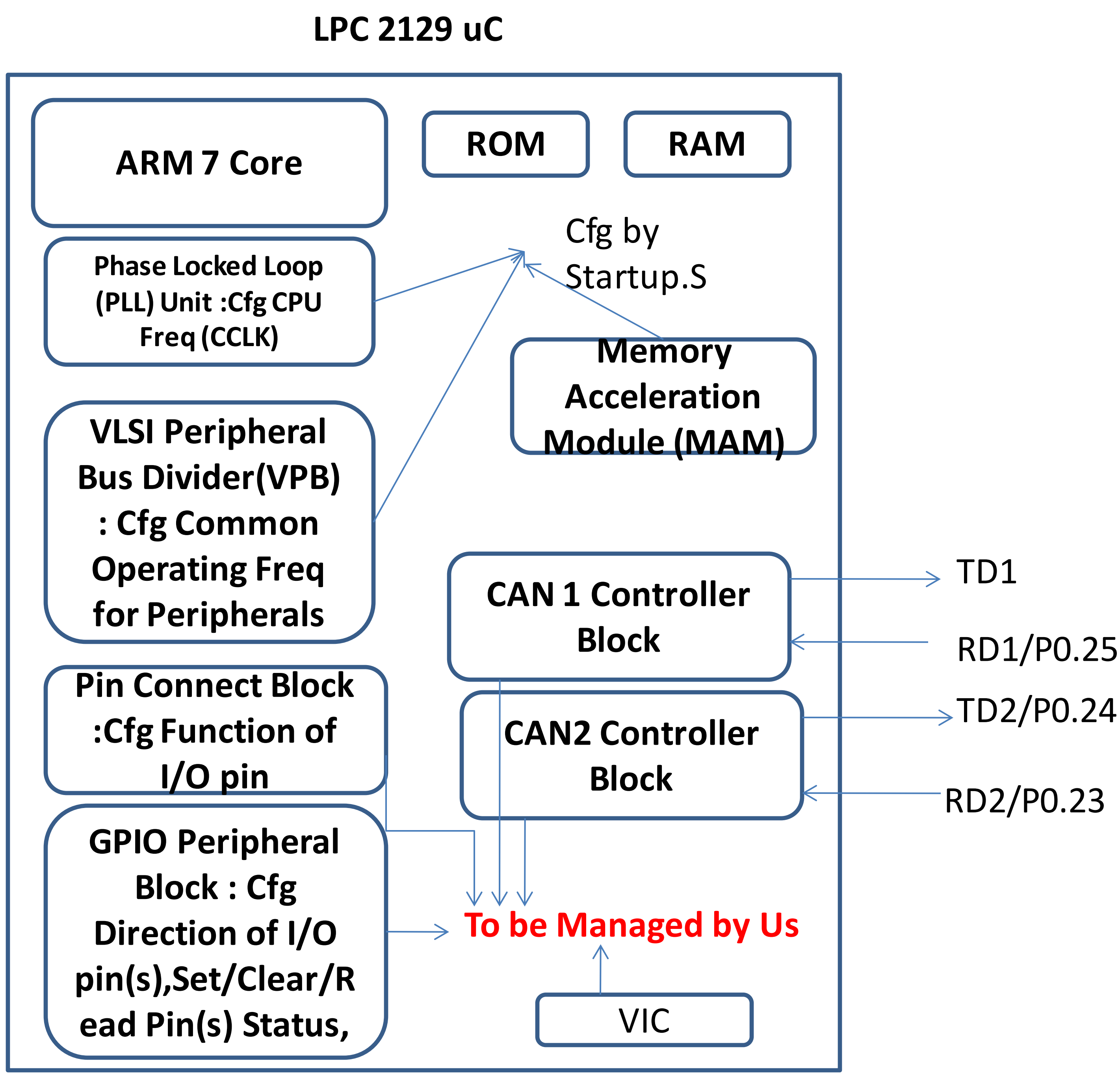
6)CPU clock up to 60 MHz,

7)On-chip crystal oscillator Circuit

8)On-chip PLL

LPC2129 Features Cont’d

9)Two UARTs(UART0 & UART1),
10)I2C serial interface,
11)2 SPI serial interfaces(SPI0 & SPI1)
12)Real Time Clock(RTC),
13)Watchdog Timer,
14)4-channels Onchip 10bit ADC,
15)Two Onchip 32-bit timers (Timer0 & Timer 1) with (7 capture/compare channels),
16)PWM unit with up to 6 PWM outputs,
17)2 CAN channels(CAN1 & CAN2).
18)Vectored Interrupt Controller with upto 32 interrupts available,


LPC 2129 uC
ARM 7 Core
ROM
RAM
Phase Locked Loop (PLL) Unit :Cfg CPU Freq (CCLK)
Cfg by Startup.S
Memory Acceleration Module (MAM)
VLSI Peripheral Bus Divider(VPB) : Cfg Common Operating Freq for Peripherals
CAN 1 Controller Block
TD1
RD1/P0.25
Pin Connect Block :Cfg Function of I/O pin
CAN2 Controller Block
TD2/P0.24
RD2/P0.23
GPIO Peripheral Block : Cfg Direction of I/O pin(s),Set/Clear/Read Pin(s) Status,
To be Managed by Us
VIC

Accessing Peripheral SFRs of LPC2129 Using Keil Embedded C

- ***<u>All peripherals for ARM are memory-mapped</u>***, they can be accessed as normal memory locations ***<u>indirectly</u>***.
- Each SFR location can be accessed as shown below:

- ***<u>#define SFR (*((volatile unsigned int *) 32-bit address))</u>***

Used in LPC21xx.h Header File

OR

- ***<u>volatile unsigned int *sfr=(unsigned int *) 32-bit address;</u>***

PIN CONNECT BLOCK

Used for Configuring/Selecting I/O Pin Functionality Through 3,32-bit SFRs namely :

- PINSEL0
- PINSEL1
- PINSEL2

GPIO BLOCK

Used If Pin Connect Block Has Configure I/O Pin(s) for GPIO function using 4 Types of 32-bit SFRs for Each Port namely P0 & P1

- IODIR0/1 — For selecting the direction (input/output pin)
- IOSET0/1
- IOCLR0/1
- IOPIN0/1 — can be used for input pin functionality

Can be used to work with output pins

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CALCULATIONS FOR CAN: SPREAD SHEET FOR QUANTA vs BRP

@PCLK 60MHz,&

@BIT_RATE 125KHz,

BRP=PCLK/(BIT_RATE*QUANTA)

BRP=(480/QUANTA) ,where 8>= QUANTA <=25

QUANTA = 8, BRP = 60

QUANTA = 10,BRP = 48

QUANTA = 12,BRP = 40

QUANTA = 15,BRP = 32

QUANTA = 16,BRP = 30

QUANTA = 20,BRP = 24

QUANTA = 24,BRP = 20

SPREAD SHEET FOR QUANTA vs BRP

@PCLK 12MHz,&

@BIT_RATE 125KHz,

BRP=PCLK/BIT_RATE*QUANTA

BRP=(96/QUANTA) ,where 8>= QUANTA <=25

QUANTA = 8, BRP = 12

QUANTA = 16,BRP = 6

QUANTA = 24,BRP = 4

Calculations For CAN

Since SAMPLE_POINT (70%) = 0.7 * QUANTA
= 0.7 * 16
= 11.2 (ignore decimal part)
= 11 (1+TSEG1)

Since NBT=1+(PropSeg+PS1)+PS2=1+TSEG1+TSEG2

TSEG1 = 10

TSEG2 = 5

if(Tseg2 >= 5 Tq)

SJW=4 Tq

if(Tseg2 < 5 Tq)

SJW =(Tseg2 - 1) Tq

CAN Controller SFRs in LPC2129 uC AtLeast

CAN 1 Controller SFRs

- C1MOD
- AMFR
- C1BTR
- C1GSR
- C1CMR
- C1TID1
- C1TFI1
- C1TDA1
- C1TDB1
- C1RID
- C1RFS
- C1RDA
- C1RDB

CAN 2 Controller SFRs

- C2MOD
- AMFR
- C2BTR
- C2GSR
- C2CMR
- C2TID1
- C2TFI1
- C2TDA1
- C2TDB1
- C2RID
- C2RFS
- C2RDA
- C2RDB

/* can.h */

```
#ifndef __CAN_H__
#define __CAN_H__

#include "types.h"

struct CAN_Frame
{
	u32 ID;
	struct BitField
	{
		u8 RTR : 1;
		u8 DLC : 4;
	}vbf;
	u32	Data1,Data2;
};

void Init_CAN1(void);
void CAN1_Tx(struct CAN_Frame);
void CAN1_Rx(struct CAN_Frame *);

#endif
```

Understanding Code Block for Initialization of

CAN Controller 1

CAN Controller 1 Initialization : (defined in can.c)*/

```
void Init_CAN1(void)
{
   //cfg p0.25 pin as CAN1_RX pin(RD1),TD1 is exclusive
   PINSEL1|=RD1_PIN; //using defines from can_defines.h
                     // #define RD1_PIN 0x00040000 ,
                     //as RD1/ (i.e CAN1_RX)/p0.25
   //Reset CAN1 controller
   C1MOD=1;
   //All received messages are accepted
   AFMR=2;
   //Set baud Rate for CAN
   C1BTR=BTR_LVAL; //using defines from can_defines.h
   //Enable CAN1 controller
   C1MOD=0;
}
```

How BTR_LVAL is Computed in can_defines.h

- //defines required for C1BTR in can_defines.h
- #define PCLK 60000000 //Hz
- #define BIT_RATE 125000 //Hz
- #define QUANTA 16
- #define BRP (PCLK/(BIT_RATE*QUANTA))
- #define SAMPLE_POINT (0.7 * QUANTA)
- #define TSEG1 ((int)SAMPLE_POINT-1)
- #define TSEG2 (QUANTA-(1+TSEG1))
- #define SJW ((TSEG2 >= 5) ? 4 : (TSEG2-1))
- #define SAM 0 //0 or 1 ,sample bus 1 or 3 time(s)
- Finally:
- **#define BTR_LVAL (SAM<<23|(TSEG2-1)<<20|(TSEG1-1)<<16|(SJW-1)<<14|(BRP-1))**

Understanding Code Block for Transmission of

CAN Frame

```c
//  defined in can.c
void CAN1_Tx(struct CAN_Frame txFrame)
{
    // Checking that the TX buffer is empty in C1GSR
     while((C1GSR&TBS1_BIT_READ)==0);
    // place 11-bit tx id in C1T1D1
    C1TID1=txFrame.ID;
    // place cfg whether data/remote frame & no of data bytes in message
    C1TFI1=txFrame.vbf.RTR<<30|txFrame.vbf.DLC<<16;
    // For Data Frame place 1 to 8 bytes data into Data Tx Buffers
    if(txFrame.vbf.RTR!=1)
    {
      //Place data bytes 1-4 in C1TDA1
      C1TDA1= txFrame.Data1;
      //Place data bytes 5-8 in C1TDB1
      C1TDB1= txFrame.Data2;
    }
     //Select Tx Buf1 & Start Xmission using
     C1CMR=STB1_BIT_SET|TR_BIT_SET;
     //monitor tx status in C1GSR
    while((C1GSR&TCS1_BIT_READ)==0);
}
```

```c
//defines in can_defines.h
//defines for C1CMR bit set
#define TR_BIT_SET   1<<0
#define RRB_BIT_SET  1<<2
#define STB1_BIT_SET 1<<5

//defines for C1GSR bit check
#define RBS_BIT_READ  1<<0
#define TBS1_BIT_READ 1<<2
#define TCS1_BIT_READ 1<<3
```

/* mainTxNode.c*/

```
#include <lpc21xx.h>
#include "types.h"
#include "can.h"

int main(void)
{
    u8 I;  struct CAN_Frame txFrame;          i=sizeof(txFrame);
    txFrame.ID=2;   txFrame.vbf.RTR=0;
    txFrame.Data1=0x34333231;     txFrame.Data2=0x38373535;

    Init_CAN1();
    while(1)
    {
        for(i=0;i<=8;i++)
        {
          txFrame.vbf.DLC=i;
         CAN1_Tx(txFrame);
        }
    }
}
```

Now
Using Keil IDE
To

Simulate CAN Communication-Transmission

Understanding Code Block for Reception of

CAN Frame

```c
//defined in can.c
void CAN1_Rx(struct CAN_Frame *rxFrame)
{
    //wait for CAN frame recv status
    while((C1GSR&RBS_BIT_READ)==0);
    //read 11-bit CANid of recvd frame.
    rxFrame->ID=C1RID;
    // read & extract data/remote frame status
    rxFrame->vbf.RTR=(C1RFS>>30)&1;
    //read & extract data length
    rxFrame->vbf.DLC=(C1RFS>>16)&0x0f;
    //check if recvd frame is data frame,extract data bytes
    if(rxFrame->vbf.RTR==0)
    {
        //read 1-4 bytes from C1RDA
        rxFrame->Data1=C1RDA;
        //read 5-8 bytes from C1RDB
        rxFrame->Data2=C1RDB;
    }
    // Release receive buffer
    C1CMR=RRB_BIT_SET;
}
```

```c
//defines in can_defines.h
//defines for C1CMR bit set
#define TR_BIT_SET   1<<0
#define RRB_BIT_SET  1<<2
#define STB1_BIT_SET 1<<5

//defines for C1GSR bit check
#define RBS_BIT_READ  1<<0
#define TBS1_BIT_READ 1<<2
#define TCS1_BIT_READ 1<<3
```

/* mainCAN_RxTest.c */

```
#include <LPC21xx.h>
#include "can.h”
main()
{
   struct CAN_Frame rxFrame;
   Init_CAN1();
    while(1)
    {
       CAN1_Rx(&rxFrame);
   }
}
```

Understanding Code Block for Debug Script for

CAN 1 Peripheral

/* can_debug.ini */

/* Note:

A debug script for CAN debugging & simulation using Keil IDE.

Can be used for representing a CAN NODE connected to the bus which sends

CAN(Standard Data/Remote) frame .

*/

```
DEFINE BUTTON "DF1","Send2CAN1(1,0,8,'A','B','C','D','D','F','H','I')"
DEFINE BUTTON "RF1","Send2CAN1(1,1,1,0,0,0,0,0,0,0,0)"

func void Send2CAN1 (unsigned int id,
                     unsigned char frameType,
                     unsigned char msgLen,
                     unsigned char byte0,
                     unsigned char byte1,
                     unsigned char byte2,
                     unsigned char byte3,
                     unsigned char byte4,
                     unsigned char byte5,
                     unsigned char byte6,
                     unsigned char byte7
                     )
{
  CAN1L=msgLen; //Length of CAN Message

 if(frameType==0)
 {
      switch(msgLen)
      {
        case 1: CAN1B0=byte0; break;
        case 2: CAN1B0=byte0; CAN1B1=byte1; break;
        case 3: CAN1B0=byte0; CAN1B1=byte1; CAN1B2=byte2; break;
        case 4: CAN1B0=byte0; CAN1B1=byte1; CAN1B2=byte2; CAN1B3=byte3; break;
        case 5:
                 CAN1B0=byte0;  CAN1B1=byte1; CAN1B2=byte2; CAN1B3=byte3;
                 CAN1B4=byte4; break;
```

```
        case 6:
                CAN1B0=byte0;  CAN1B1=byte1; CAN1B2=byte2; CAN1B3=byte3;
                CAN1B4=byte4; CAN1B5=byte5; break;
        case 7:
                CAN1B0=byte0;  CAN1B1=byte1; CAN1B2=byte2; CAN1B3=byte3;
                CAN1B4=byte4; CAN1B5=byte5; CAN1B6=byte6; break;
        case 8:
                CAN1B0=byte0;  CAN1B1=byte1; CAN1B2=byte2; CAN1B3=byte3;
                CAN1B4=byte4;  CAN1B5=byte5; CAN1B6=byte6; CAN1B7=byte7;
                break;
      }
  }
  CAN1ID = id; //Send CAN 11-bit ID
  if(frameType==0)
      CAN1IN = 1;//standard data frame
  else if(frameType==1)
      CAN1IN = 3;//standard remote frame

  CAN1IN = 5;//Start Bus Activity
}
```

Now
Using Keil IDE
To

Simulate CAN Communication-
Reception