I2C

LPC1768 I2C Registers

- I2CONSET, I2CONCLR
    - Setting and clearing I2C flags/bits.
    - START, STOP, ACK, EN, SIF
    - SIF should be cleared after transfer.
- I2SCLL, I2SCLH
    - Clock divider for I2C clock.
    - I2C freq = PCLK / (I2SCLL + I2SCLH)
    - I2C freq should be 100KHz, 400KHz or 1MHz as per slave datasheet.
    - I2SCLL + I2SCLH values also decide duty cycle.
- I2DAT
    - Data Register for transmit as well as receive
- I2STAT
    - 5 bits I2C status
    - I2C is a state machine. After each step, 5-bit status code is produced.
- I2ADDR
    - I2C address of LPC1768 when it is used as slave on I2C bus.

I2C Bus Arbitration

- When two devices send START bit at the same time, bus arbitration occur.
- In this case, whichever device send 0 bit first will win the arbitration and get ownership of the bus. The other device abort the data transfer and wait.
- Each I2C have level sensor cct for data line. It always check the line voltage level after writing bit to it. If voltage differs from the transmitted bit (i.e. bit transmitted is 1, but voltage level is 0), the device loose arbitration.

I2C Clock Stretching

- When receiver is busy in processing data, it holds clock line low. Due to Wired-AND cct, effective clock line will be low.
- As no clock present, the data transfer is not possible.
- This mechanism is used to synchronise between sender and receiver.

I2C errors

- Write collision
    - Transmitter send next byte of data before current byte is transmitted.

- Read collision
    - New data byte is received before reading current data byte.

ADC

- Used to convert analog data into digital value.
- This analog data is available from various sensors, mic, ...
- ADC types
    - Flash ADC
    - Dual Slope ADC
    - Successive Approximation ADC
- Successive Approximation ADC
    - Implemented in most of controllers
    - It internally use DAC and comparator to get difference between approximated value and actual value. Based on difference, it does next approximation.

LPC1768 ADC Registers

- ADCR
    - bits[7:0] for select ADC channel e.g. for CHN3, bit3 = 1.
    - bits[15:8] for CLKDIV
        - ADC clock = PCLK / (CLKDIV+1)
        - Max ADC clock should be 13 MHz.
        - If PCLK=18MHz, CLKDIV=1, so that ADC clock = PCLK/(1+1) = 9 MHz.
    - bit16: burst mode (continous conversions)
    - bit21: Power Down (0) and On (1).
    - bit24: To start conversion set it 1.
- ADGDR
    - bits[15:4] for digital output
    - bits[26:24] for channel number
    - bit31 -- conversion completed.
    - bit30 -- reading overrun (new reading is recorded before current reading is read).
- ADDRx
    - bits[15:4] for digital output
    - bit31 -- conversion completed.
    - bit30 -- reading overrun (new reading is recorded before current reading is read).

Blueboard Cct

- Potentiometer --> P0.25 (AD0.2)

Programming

- adc_init()
    - Make P0.25 as input
    - enable ADC AD0.2
    - Config ADC -- ADCR
        - Select Channel 2
        - CLKDIV = (2-1)
        - Enable Power
- adc_read()
    - start conversion (ADCR -- START bit)
    - wait for conversion (ADGDR -- DONE bit)
    - read the result (ADGDR)

CM3 Instruction Set

- Thumb2 Instruction Set
    - 16-bit Instructions + 32-bit Instructions

CM3 Assembly Programming

- Assembly code (.s) --> Assembler --> Object code (.o) --> Linker --> Executable code (.elf) --> ObjCopy --> Binary code (.bin)
- Binary code (.bin) --> ARM Emulator <-- Debugger
- ARM toolchain tools
    - Assembler: arm-none-eabi-as
    - Linker: arm-none-eabi-ld
    - ObjCopy: arm-none-eabi-objcopy
    - Debugger: arm-none-eabi-gdb
- ARM Emulator
    - QEmu is open-source hardware Emulator for multiple architectures e.g. x86, ARM, SPARC, ALPHA, PPC, MIPS, ...
    - Android Emulator is based on QEmu.
    - qemu-system-arm is ARM Emulator.
    - terminal> qemu-system-arm -machine help
    - lm3s6965evb* and lm3s811evb are CM3 based controllers.
- basic.s
    - Vector table
    - Reset Handler (_start)
    - main()
    - Default Handler

- Compilation
  - Assembly code (.s) --> Assembler --> Object code (.o)
    - terminal> arm-none-eabi-as -mcpu=cortex-m3 -march=armv7 -mthumb -gwarf2 -o basic.o basic.s
  - Object code (.o) --> Linker --> Executable code (.elf)
    - terminal> arm-none-eabi-ld -T./lm3.ld -o basic.elf basic.o
  - Executable code (.elf) --> ObjCopy --> Binary code (.bin)
    - terminal> arm-none-eabi-objcopy -O binary basic.elf basic.bin
- Execution (Debugging)
  - terminal> qemu-system-arm -machine lm3s6965evb -cpu cortex-m3 -S -gdb tcp::1234 -nographic -kernel basic.bin
    - Runs basic.bin into ARM Emulator and suspend exection (-S). Also debugging is available on port 1234.
  - terminal> arm-none-eabi-gdb basic.elf -ex "target remote localhost:1234"
    - Start the debugger and execute the command "target remote localhost:1234". It connect to qemu on port 1234 and start debug step by step.
  - terminal> pkill qemu-system-arm
    - stop qemu.
  - Debugger commands
    - break label -- set breakpoint to label
    - step -- go into the function
    - next -- execute function and go to next step.
    - cont -- run to next breakpoint
    - info registers r0 r1 r2 cpsr ... -- to print register values
    - quit -- stop debugging

Jump Instructions

- B label
  - Branch/Jump Instruction
  - The address given by label is loaded in PC.
- BL label
  - Function call Instruction
  - The address of next Instruction is copied into LR.
  - The address given by label is loaded in PC.
- MOV PC, LR

    - Return from function.
    - Copies LR into PC.

Data processing Instructions

- Data movement Instructions
    - MOV, MVN
- Arithmetic Instructions
    - ADD, SUB, ADC, SBC
- Multiply Instructions
    - MUL, MLA
- Barrel shifter Instructions
    - LSR, ASR, LSL
- Logical Instructions
    - AND, OR, EOR
- Comparision Instructions
    - CMP, CMN