Agenda

- Revision
- Interrupt vs Polling
- UART
    -
    -
- Timers

Interrupt

- CPU is executing the user code. Peripheral send interrupt signal to the CPU.
- CPU pause the current execution, get the address of ISR from the vector table and call it.
- ISR execution is completed. Then paused user program continue to execute.

Interrupt programming in CM3 (using CMSIS)

- step1: Implement the ISR and place it in appropriate slot in vector table.
    - Implement ISR function with same prototype as declared in startup.c file.
        - void EINT2_IRQHandler(void);
    - EINT2_IRQHandler() is already placed in right slot in vector table. However that function is declared as weak (**attribute((weak))**) i.e. when not defined by user, the default function will be taken (#pragma weak).
- step2: Enable interrupt in NVIC.
    - NVIC_EnableIRQ(EINT2_IRQn); (fn is declared in core_cm3.h)
- step3: Enable interrupt in peripheral.
    - Peripheral specific.
    - For example (EINT2):
        - EXTMODE |= BV(2);
        - EXTPOLAR |= BV(2);
        - EXTINT |= BV(2);

Compiler Optimization

- Compiler always (depends on settings) try to generate optimized code, so that it can execute faster or take smaller space in program memory.
- gcc Compiler Optimization levels: 0, 1, 2, 3, s
    - s is Optimization for size (generate compact code).
    - 0...3 is Optimization for speed.

        - 0 is no Optimization.
        - 3 is highest Optimization.

    - Example: If variable is modified in a loop repeatedly, compiler may copy it into register and modify from there (instead of modifying each time in RAM, which is slower). At the end of loop, it will write register value back to the RAM location.
    - However, if variable is getting modified outside current execution context (loop), then the changes done in RAM location will not visible in that context (loop). This may lead to unexpected output. Such situations Optimization need to be disabled.

        - To disabled Optimization for entire program: -O0
        - To disabled Optimization for that variable: volatile

Interrupt vs Polling

- CPU keep checking for the event continuously, is referred as polling. CPU register for the event and when event occured, it is notified to CPU, this mechanism is called as interrupt.
- e.g. staircase -- elevator.
- In polling, CPU is busy in checking and hence cannot do any other task. In interrupt, CPU can continue doing other task and execute ISR when interrupt occur.
- In polling, CPU is waiting for IO to be completed; this is called as "Sync IO". In interrupt, CPU is not waiting (doing other task) while IO is in progress; this is called as "Asyn IO".
- Usually polling programming is simpler than interrupt programming.

UART (RS-232)

- UART is a serial communication.
- Serial vs Parallel communication

    - In Parallel, multiple wires are connected to transfer data, so multiple bits can be transferred at a time. In serial, only single wire is used for data transfer (in one direction), so can transfer only one bit at a time.
    - Typically, serial protocol cct have shift register to transfer data. These shift registers operate on same frequency clock.
    - Parallel communication have better speed than serial. But recent development, serial communication speed in quiet better.

- Serial communication types

    - Simplex
    - Half duplex
    - Full duplex

RS-232 protocol

- Physical characteristics

    - RS-232 is peer-to-peer communication mechanism (it is not bus protocol)
    - 3-wire protocol: Rx, Tx and Gnd.

        - It follows CMOS levels
            - 1: -3V to -25V
            - 0: +3V to +25V
        - Typically MAX232 IC is used for converting TTL levels to CMOS levels. MAX232 is also referred as UART line driver.
        - RS-232 speed measured in baud rate. Baud rate is defined as number of signal changes in a second. For conductor material, it is same as bps (bits per second). The common baud rates are: 9600, 38400, 115200, ...
- Logical characteristics
        - Data frame
            - Start bit -- Always 0
            - 8 Data bits
            - Parity bit (optional) -- Even or odd
            - Stop bits (1 or 2) -- Always 1
- Serial connector
        - DB25 (old)
        - DB9 (current)
            - 2: Rx
            - 3: Tx
            - 5: Gnd
            - Remaining pins are handshaking signals for modem communication.
        - Half serial cable
            - Only 3 wires are connected
        - Full serial cable
            - All 9 wires are connected
- Nowadays serial ports are not common for desktops/laptops. But we use USB to Serial converter (PL2303 is common).

UART registers in LPC1768

- RBR: Receive Buffer register
- THR: Transmit Holding register
- DLL & DLM: Divisor Latch Least Byte & Most Byte. Used to configure baud rate.
- LCR: Line Control register -- to configure UART
        - bit [1:0] = Data Length = 11 for 8-bit
        - bit 2 = stop bits = 1(0) or 2(1) stop bits
        - bit 3 = parity enable(1) or disable(0)
        - bit 7 = DLAB

            - 1: can read/change DLL & DLM
            - 0: cannot read/change DLL & DLM

- LSR: Line Status register -- to monitor UART
    - bit 0 = RDR (Recr data ready)
        - 0: Data not available
        - 1: Data available
    - bit 5 = THRE (Tx holding Regr empty)
        - 0: THR not empty (last data transfer is in progress).
        - 1: THR empty (last data transfer is completed).
- FCR: FIFO Control register -- to control FIFO
    - bit 0 = Enable (1) or disable (0) FIFO
- PINSEL0
    - bits [1:0] & [3:2] = 10 for UART3
    - bits [5:4] & [7:6] = 01 for UART0

UART cct

- U0 --> ~~P2.3 (Tx) & P2.5 (Rx)~~ --> P0.2 & P0.3 (UART0)
- U1 --> ~~P3.1 (Tx) & P3.2 (Rx)~~ --> P0.0 & P0.1 (UART3)
- UART0 is given for ISP (in system programming) i.e. to program (burn) LPC1768. Can also be used for communication.
- UART3 can be used for communication.

UART baud rate calculation

- baud = PCLK / (16 * dl) = PCLK / 16 / dl = (PCLK >> 4) / dl
- dl = PCLK / (16 * baud) = PCLK / 16 / baud = (PCLK >> 4) / baud
- Example: Baud=9600 (PCLK=18MHz)
    - dl = 18000000 / 16 / 9600

UART programming

- uart_init(baud)
    - use P0.0 & P0.1 as UART3 pins - PINSEL0
    - config UART - 8-bits, No parity, 1-stop bits - LCR
    - calculate dl - dl = (PCLK>>4)/baud
    - set dl - DLL & DLM (to set ensure DLAB=1)
- uart_getch()

    - check if data is available to read (LSR) & wait if not.
    - read the data (RBR)

- uart_putch()
    - check if last data is transferred (LSR) & waif if not.
    - write the data (THR)

- uart_gets()
    - read each char (using uart_getch()) until '\r' and append in a string.

- uart_puts()
    - send each char till '\0' (using uart_putch()).

UART errors

- Visible in LSR (bits).
- Frame Error: When STOP bit is 0.
- Read Overrun: If next data byte is received before current data is read. In this case current data byte is lost.
- Parity Error: Parity mismatched (as of config) while receving data.

Timer

- Timer vs Counter
    - Timer is a counter that counts clock cycles of fixed frequency, so that based on number of clock cycles time can be calculated.

- If timer clock is TCLK, then clock period is 1/TCLK.
- If timer counter counts n cycles, then time measured is n * (1/TCLK).

LPC1768 timers

- Most of the timers are 32 bit.
- Timer0/1/2/3 *
- Watchdog Timer *
- SysTick Timer *
- Repetitive Interval Timer
- Real Time Clock *

Timer0/1/2/3

- All these timers are of 32-bit.
- TCR
    - bit0: Enable(1) or Disable(0)
    - bit1: Reset(1) and 0 to start counting.

- CTCR

    - bits[1:0] = 00 for Timer mode
- MCR
    - for each MRx, three bits I, R, S.
- IR
    - 4 bits for MRx and 2 bits for CRx

Timer programming

- Write a delay function for delay of given ms.
- timer_init()
    - enable timer and reset it.
    - timer mode.
    - set prescalar (18)
- timer_delay_ms(ms)
    - reset the timer
    - calculate count based on timer.
    - Set value in MR0.
    - Set MCR -- stop timer and generate interrupt.
    - start the timer
    - wait for interrupt flag in IR