

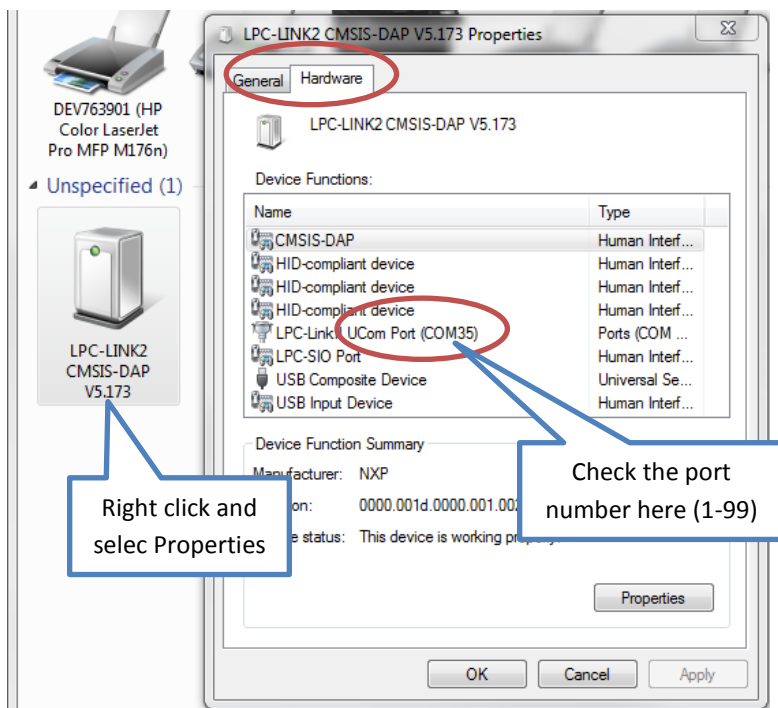
Morse code transmitter

How to use putty to communicate with the board

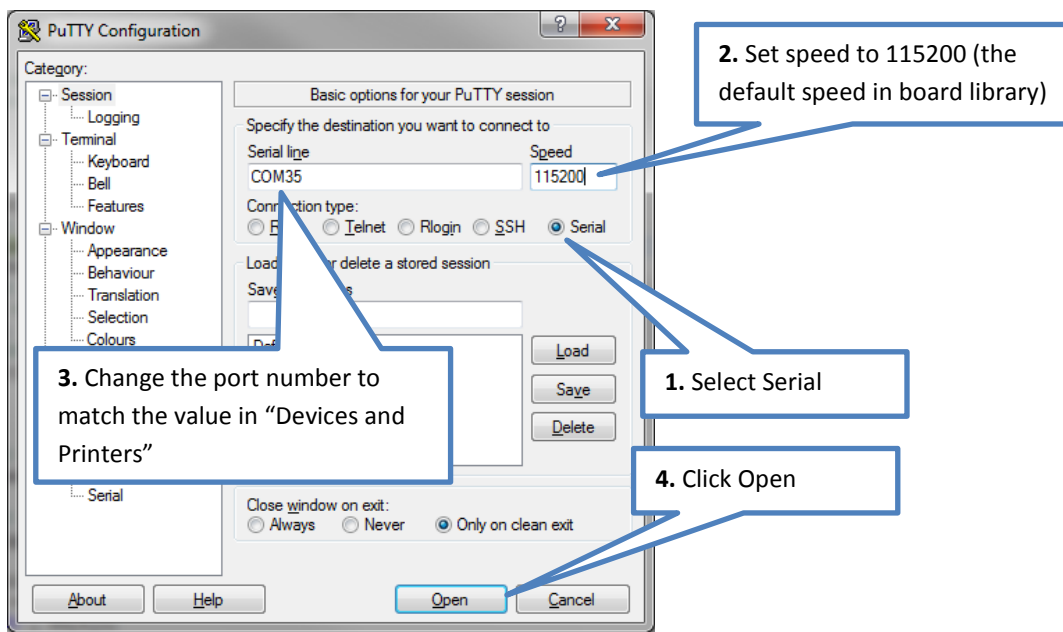
When you plug in your LPCXpresso board the drivers are installed automatically. The debugger part of the board contains USB-UART that sends the data from UART0 to your computer. The USB-UART shows up as COM-port on your PC. The USB-UART does not usually show up until you start debugging. On the first debugger session LPCXpresso IDE sends the debugger code to the board. The debugger code is retained in the memory until you unplug the board.

The port number can be anything between 1 and 99 depending on the USB port (and luck...). Usually the COM port number stays the same if you use same board and same USB port but it may change after windows (or other) update.

To find out the port number open “Devices and printers”, right click LPC-LINK2 CMSIS-DAP, select Properties and navigate to Hardware tab.



Start PuTTY (download from putty.org) and do the following steps:



Try out the connection with the following program. Make sure that all initialization code created by the wizard is executed before your own code.

```
int main(void)
{
    #if defined (__USE_LPCOPEN)
        // Read clock settings and update SystemCoreClock variable
        SystemCoreClockUpdate();
    #if !defined(NO_BOARD_LIB)
        // Set up and initialize all required blocks and
        // functions related to the board hardware
        Board_Init();
        // Set the LED to the state of "On"
        Board_LED_Set(0, true);
    #endif
    #endif

    Board_UARTPutSTR("\r\nHello, World\r\n");
    Board_UARTPutChar('!');
    Board_UARTPutChar('\r');
    Board_UARTPutChar('\n');
    int c;
    while(1) { // echo back what we receive
        c = Board_UARTGetChar();
        if(c != EOF) {
            if(c == '\n') Board_UARTPutChar('\r'); // precede linefeed with carriage return
            Board_UARTPutChar(c);
        }
    }
    return 0;
}
```

A simple delay function

Systick timer and a function that generates a fairly accurate delay. Systick timer is assumed to run at 1000 Hz (once per millisecond). See `periph_blinky` project for example on configuring the systick timer. Systick timer must be configured before you can call `Sleep()`.

You can add this to your project if you need to generate delays.

```
static volatile int counter;

#ifdef __cplusplus
extern "C" {
#endif
/**
 * @brief   Handle interrupt from SysTick timer
 * @return  Nothing
 */
void SysTick_Handler(void)
{
    if(counter > 0) counter--;
}
#ifdef __cplusplus
}
#endif

void Sleep(int ms)
{
    counter = ms;
    while(counter > 0) {
        __WFI();
    }
}
```

`SysTick_Handler` is called by the timer hardware at the frequency specified by the systick timer configuration. Note that you must not call `SysTick_Handler` from your program and that the name of the function must be spelled exactly as in the example above. The `#ifdef __cplusplus` is an essential part of the code and must not omitted.

The delay generation is simple – just call `Sleep` with the required delay time as parameter. For example to generate 200 ms delay do `Sleep(200)`.

Note: Hardware takes care of calling `SysTick_Handler`. Do not call `SysTick_Handler` in your code. Do not modify the handler code or `Sleep()`.

Exercise 1

Write a program that reads characters from UART and echoes them back. Before sending the letters back the program changes the letters to either upper case or lower case. The mode of operation is selected with the onboard button. The mode toggles every time when you press the button. Hint: use the onboard led to indicate mode (to uppercase/to lowercase) for easier operation.

Your program must use the IO-pin object from lab2 to read the state of the button.

Exercise 2

Write a program that reads characters from UART until linefeed is pressed or 80 characters have been read, whichever occurs first. The program then sends Morse coded text with the red led and pin A0 (port 0, pin 8). Red led is for visual inspection and the IO-pin is for morse code decoder.

Use ITU Morse code for the text. Replace characters that don't have a Morse code symbol with 'X'.

International Morse code is composed of five elements:

1. short mark, dot or "dit" (·) — "dot duration" is one time unit long
2. longer mark, dash or "dah" (–) — three time units long
3. inter-element gap between the dots and dashes within a character — one dot duration or one unit long
4. short gap (between letters) — three time units long
5. medium gap (between words) — seven time units long

For more details see Wikipedia article: http://en.wikipedia.org/wiki/Morse_code

```
const int DOT = 1;
const int DASH = 3;
struct MorseCode {
    char symbol;
    unsigned char code[7];
};
const MorseCode ITU_morse[] = {
{ 'A', { DOT, DASH } }, // A
{ 'B', { DASH, DOT, DOT, DOT } }, // B
{ 'C', { DASH, DOT, DASH, DOT } }, // C
{ 'D', { DASH, DOT, DOT } }, // D
{ 'E', { DOT } }, // E
{ 'F', { DOT, DOT, DASH, DOT } }, // F
{ 'G', { DASH, DASH, DOT } }, // G
{ 'H', { DOT, DOT, DOT, DOT } }, // H
{ 'I', { DOT, DOT } }, // I
{ 'J', { DOT, DASH, DASH, DASH } }, // J
{ 'K', { DASH, DOT, DASH } }, // K
{ 'L', { DOT, DASH, DOT, DOT } }, // L
{ 'M', { DASH, DASH } }, // M
{ 'N', { DASH, DOT } }, // N
{ 'O', { DASH, DASH, DASH } }, // O
{ 'P', { DOT, DASH, DASH, DOT } }, // P
{ 'Q', { DASH, DASH, DOT, DASH } }, // Q
{ 'R', { DOT, DASH, DOT } }, // R
{ 'S', { DOT, DOT, DOT } }, // S
{ 'T', { DASH } }, // T
{ 'U', { DOT, DOT, DASH } }, // U
{ 'V', { DOT, DOT, DOT, DASH } }, // V
{ 'W', { DOT, DASH, DASH } }, // W
{ 'X', { DASH, DOT, DOT, DASH } }, // X
{ 'Y', { DASH, DOT, DASH, DASH } }, // Y
{ 'Z', { DASH, DASH, DOT, DOT } }, // Z
{ '1', { DOT, DASH, DASH, DASH, DASH } }, // 1
{ '2', { DOT, DOT, DASH, DASH, DASH } }, // 2
{ '3', { DOT, DOT, DOT, DASH, DASH } }, // 3
{ '4', { DOT, DOT, DOT, DOT, DASH } }, // 4
{ '5', { DOT, DOT, DOT, DOT, DOT } }, // 5
{ '6', { DASH, DOT, DOT, DOT, DOT } }, // 6
{ '7', { DASH, DASH, DOT, DOT, DOT } }, // 7
{ '8', { DASH, DASH, DASH, DOT, DOT } }, // 8
```

```

{ '9', { DASH, DASH, DASH, DASH, DOT } }, // 9
{ '0', { DASH, DASH, DASH, DASH, DASH } }, // 0
{ 0, { 0 } } // terminating entry - Do not remove!
};

```

The character at index k can be accessed with `ITU_morse[k].symbol`. If the character is zero then you are at the end of the morse code table.

Code is an array of non-zero number that indicate the length of the marks in the code. To read the code at index k: start with `i=0`, loop through `ITU_morse[k].code[i]` increasing `i` on each round until you see a zero that marks the end of the code.

This must implement morse code sender class. The constructor takes two pointers to IO-pin object as parameters: one for led and one for decoder output pin. There must be public methods for sending out a string in morse code. Use method overloading to implement both string class and C string interfaces.

The functionality of this program will be tested against a Morse code decoder. Do not use dot length that is shorter than 10 ms. Dot duration of 10 ms is about 100 wpm which is way too fast for a human to decode from led blinks. Start with slower speed and test first with easy words like SOS.

Exercise 3

Improve your program so that the time unit is configurable. Program must have the following commands:

`wpm <number>`

- Set transmission speed in words per minute. Assume that average word length is 60 time units long.

`send <text to send>`

- Send text that follows the command

`set`

- Print current settings (wpm and dot length in milliseconds)