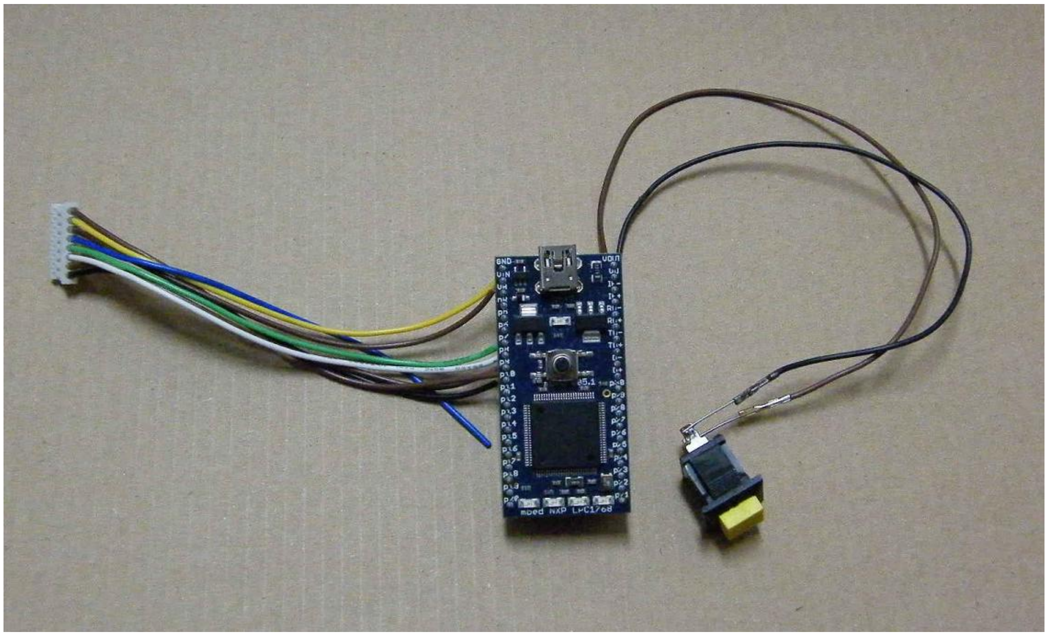
NXP mbed Design Challenge

Registration Number: NXP3870

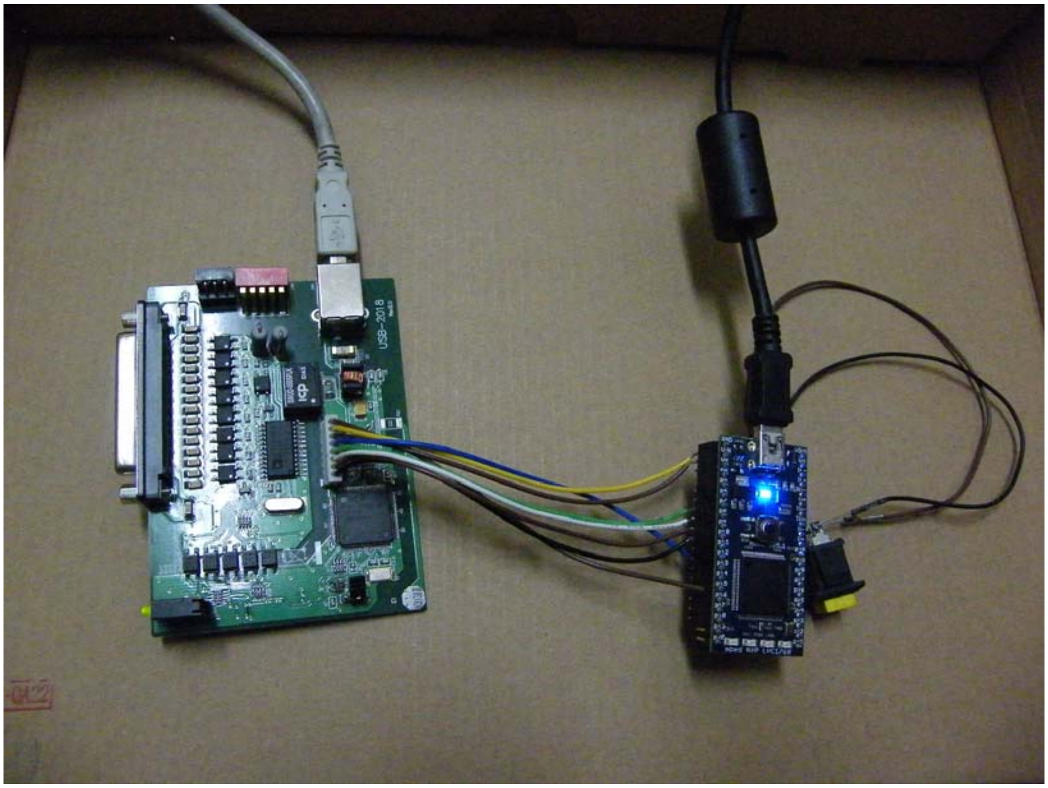
LPC17xx ISP Programmer



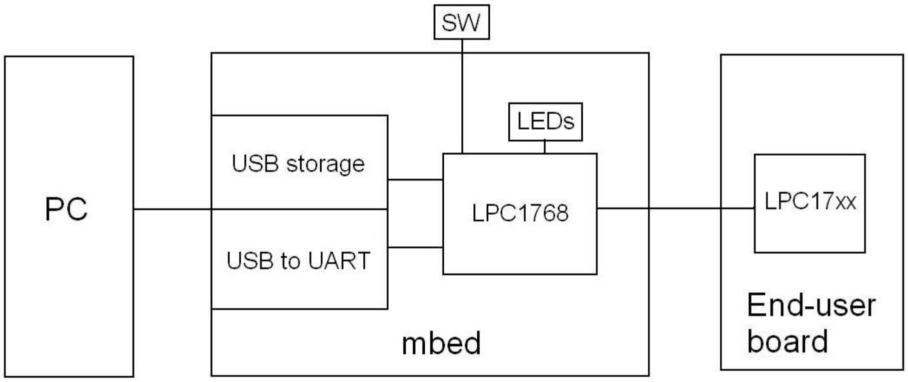
The on-chip flash memory of the LPC17xx can be programmed using its built-in boot loader when it resides in the end-user board. The programming is through UART, so a TTL to RS-232 converter is required. Furthermore, the RS-232 port is rare in modern desktop and laptop PCs. That means, you need a USB to RS-232 converter, too.

This project is to develop a device that uses **mbed** with some wiring to provide LPC17xx flash programming capability. It is connected through the USB port, which can be found in most of the desktop and laptop PCs. Besides, it is a USB to UART converter, which can be very helpful in debugging user codes executed on LPC17xx. The device is useful in developing and prototyping end-user board with LPC17xx.

The following photo shows the device connected to an end-user board.



The **mbed** is used as an USB drive and an USB to UART converter. The 4 LEDs on **mbed** are used to indicate device status. The block diagram is shown below.



Block Diagram

There are two ways to program the LPC17xx using the device.

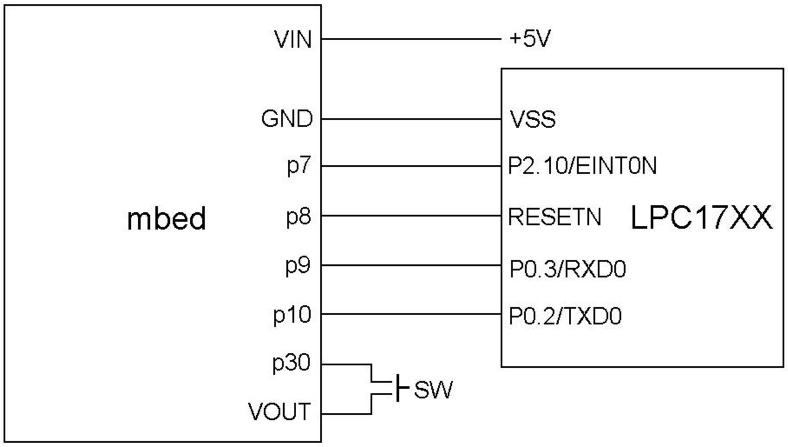
Method I

Use the Flash Magic utility, http://www.flashmagictool.com/, and just follow the normal operation steps.

Method II

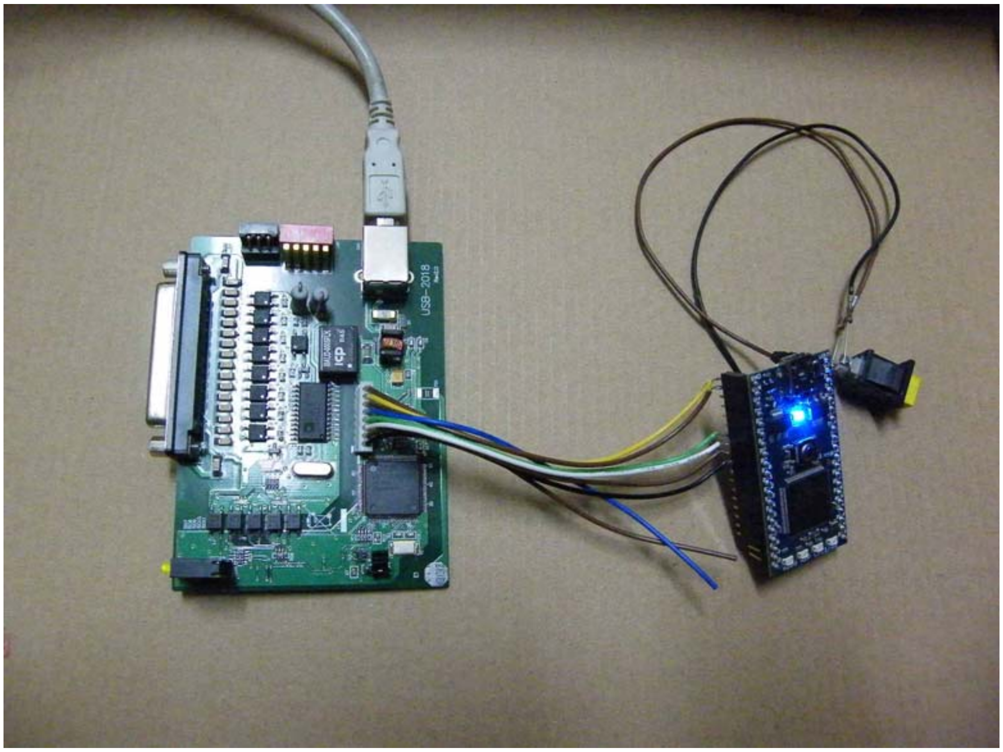
Copy the user codes in HEX file format to the device. Then, press the external button. In this way, it is easy to duplicate the user codes to other LPC17xx by just connecting the device and pressing the external button.

The wiring is very simple as shown below.



Schematic Diagram

The switch button is required only for Method II. The +5V source is required only when you want to use Method II to duplicate user codes to other LPC17xx without connecting the device to PC. The following photo shows the device connected to an end-user board and the device is not connected to PC.



Sample Codes

The following codes are used to parse the HEX file, calculate the size of binary data, and get the last sector number to be programmed.

```
int CheckHexFile(char *filename)
{
	int error = 0, reclen, addr, rectype, wChkSum, lastaddr=0, i;
	FILE *fp;
	char buf[128];

	fp = fopen(filename, "r");
	if (fp == NULL) {
		return -1;			// no file found
	}

	while (fgets(buf, sizeof(buf), fp) != NULL)
	{
		if ( buf[0] != ':' )
		{	// invalid start of a line
			error = -1;
		}
		reclen = StrToHex(buf+1,2);			// byte count
		addr = StrToHex(buf+3,4);			// address
		rectype = StrToHex(buf+7,2);			// record type
		if ( rectype == 4 )
		{
			if ( addr != 0 || reclen != 2 )
				// invalid Extended Linear Address Record
				error = -1;
		}
		else if ( rectype == 5 )
		{
			if ( addr != 0 || reclen != 4 )
				// invalid Extended Linear Address Record
				error = -1;
		}
		else if ( rectype == 1 )
		{
```

```
            if ( addr != 0 || reclen != 0 )
                // invalid End of File Record
                error = -1;
        }
        else if ( rectype != 0 )
        {
            // invalid Record
            error = -1;
        }
        wChkSum = 0;
        for ( i = 0 ; i < (reclen + 5) ; i++ )
            wChkSum += StrToHex(buf+1+i*2, 2);
        if ( (wChkSum & 0xFF) != 0 )
            // invalid checksum
            error = -1;
        if ( rectype == 0 )
        {   // find the address of last data byte
            if ( lastaddr <= (addr + reclen) )
                lastaddr = (addr + reclen);
            else
                error = -1;          // the data should be in sequence
        }
    }
    fclose(fp);
    if ( error == 0 )
        return GetLastSector(lastaddr);
    return -1;
}

// get the last sector number to be programmed
int GetLastSector(int lastaddress)
{
    if ( lastaddress <= 0xFFFF )
        return (lastaddress)/4096;          // 4KB/sector if address <= 0xFFFF
    return (lastaddress)/32768-2+16;        // 32KB/sector if address > 0xFFFF
}
```