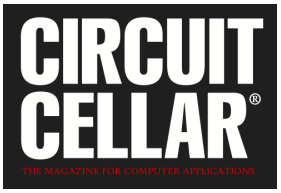


Circuit Cellar - WIZnet iMCU Design Contest 2010
Project Number: **003189**
WIZnet Components: **W7100** in iMCU7100EVB evaluation board
Microcontroller: WIZnet W7100 (8051 core)

Abstract

m7100os

A Network Operating System for the WIZnet W7100 Internet Enabled CPU

ABSTRACT

The m7100os is a Network Operating System for the WIZnet W7100 CPU. As a Network Operating System the m7100os implements most of the features of a multitasking kernel while including embedded support for a POSIX like TCP/IP layer interface to easily implement or port network enabled applications. The m7100os Operating System supports a variable number of tasks with preemption, semaphores, signal passing, message passing, software Real Time Clock (RTC), tasks priorities but it also includes a fully reentrant interface for socket programming. The OS has several TCP/IP modules implemented which can be enabled or disabled as options. These include DHCP client, simple Network Time Protocol (SNTP), interrupt based serial port driver, DNS client, IP Ping and a simple Job Scheduler. To facilitate debugging, the m7100os also has a kernel based monitor/debugger which can be called from user code or through a interrupt pin. The monitor can check the state of the different tasks and their registers, modify memory, and start the program at any location.

With the m7100os any W7100 programmer could implement a complex network enabled project without having to dedicate time to the native W7100 TCP/IP interface and with the added benefit of a reentrant network interface which simplifies porting of existing applications. The multitasking socket programming largely simplifies the development phase.

The m7100os has several different components:

- The kernel (m7100os.c), which implements the multitasking, preemtable kernel for the 8051 core of the W7100. It also provides the RTC, and kernel communications (semaphores, message passing, signal passing, etc.). The kernel uses an optimized task swapping code to be able to support the variable number of tasks and the code reentrancy. The user tasks stacks are

moved to the “external” (to the core) 64K memory space avoiding the limitations of the small 8051 stack. Each user task can have about 170 bytes of stack in the current implementation.

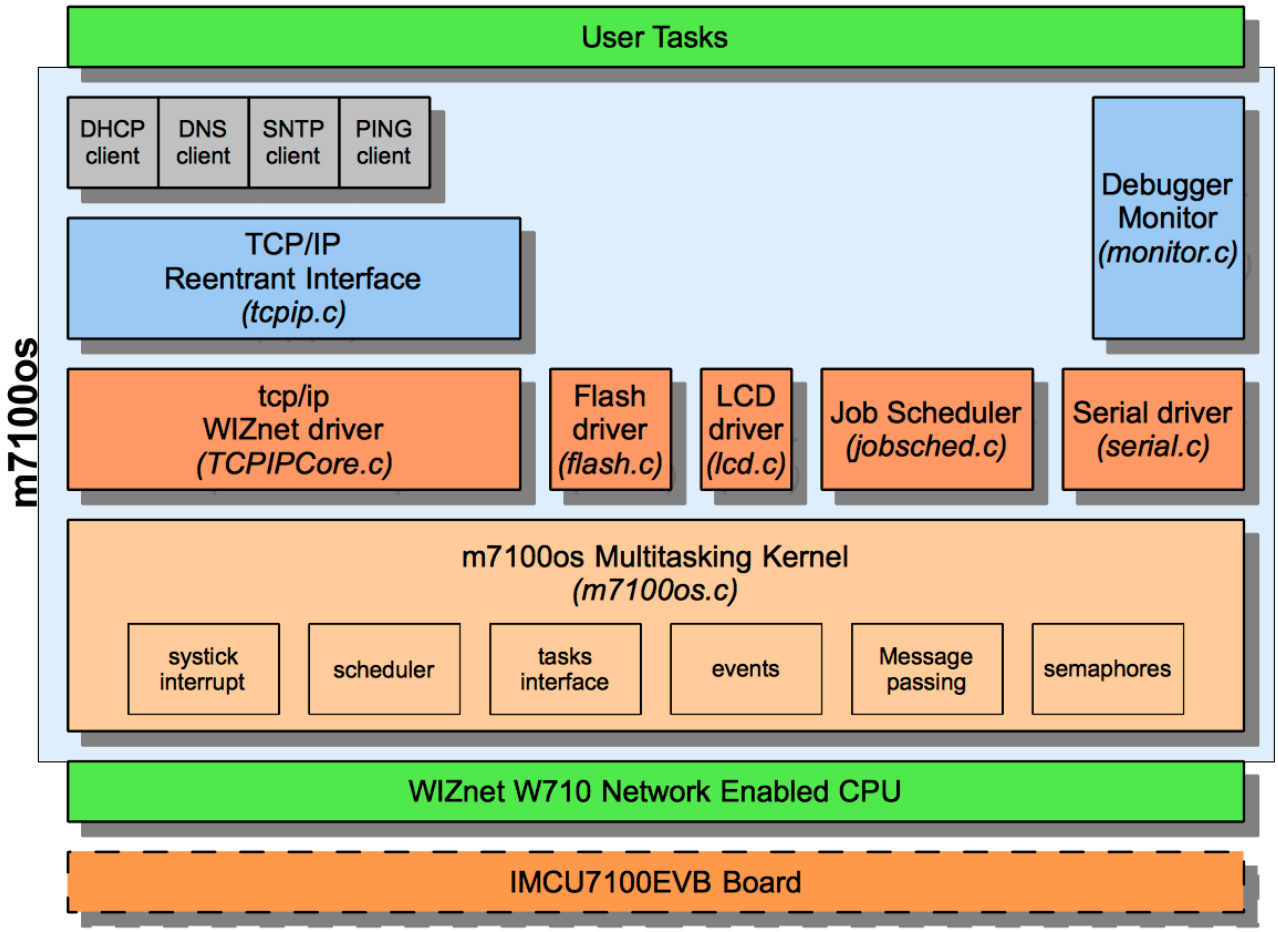
- A kernel debugger/monitor (monitor.c), which provides a software based entry call and an interrupt based hardware break point through an external pin.
- A POSIX alike (mostly) socket interface (tcpip.c), which together with the WIZnet provided (and modified) TCPIPCore.c provides the raw, ip, udp and tcp protocol interfaces to the user tasks. This interface also defines blocked and non blocked I/O with a user defined timeout.
- A flash memory interface (flash.c), to support the flash memory in the W7100 from any user task.
- A reentrant LCD interface (lcd.c), to control the evaluation board display.
- A reentrant serial port interface (serial.c), for debugging and user serial interface. The serial port is used for the monitor, but it can be used in the user code for any communication while not in the monitor.
- Several higher level TCP modules: DNS client (dns.c), Job Scheduler (jobsched.c), ICMP ping (ping.c), Simple Network Time Protocol (SNTP, to synchronize the software RTC with an external NTP server).

The m7100os submitted code is fully functional in its version 1.1. Several test samples demonstrating its capabilities are submitted with the source code. The full documentation has the details of how to compile and run the samples. This project is a software only entry and runs in the evaluation board without any modification.

The m7100os was programmed from scratch specifically for the W7100 CPU, using the open source SDCC compiler (version 2.9.0) in a Debian Linux 5.0 environment and tested with the iMCU7100EVB and the WIZnet firmware loader running in Windows.

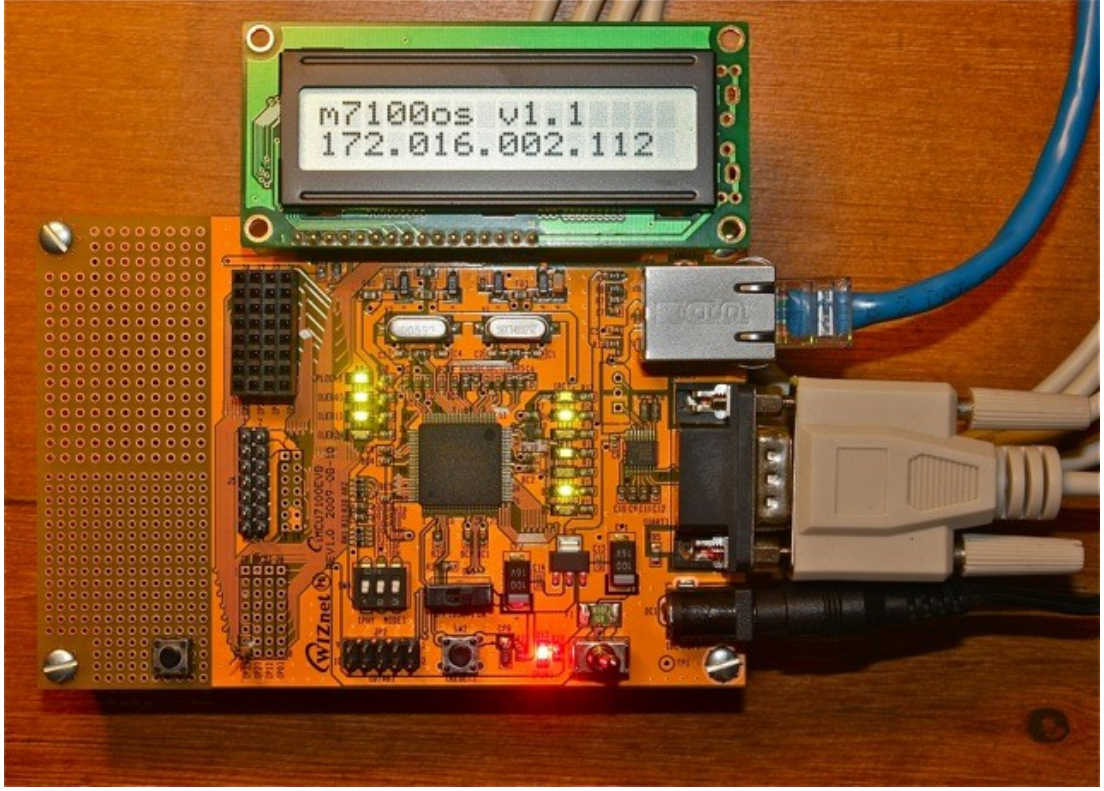
m7100os Block Schematic

The m7100os is divided in modules. The following Schematic pictures how those modules interact. The different modules can be enabled or disabled in the m7100os.h header file. The makefile of the project also have to be edited to reflect those changes.



m7100os Schematic

The m7100os project is a software only entry so no schematic is included for this contest. The sample code runs in the iMCU7100EVB evaluation board unmodified. As an option a push button can be connected to the Interrupt 2 pin and ground (nINT2, but customizable) to enable the hardware break point to enter directly to the m7100os debugger/monitor.



Prototype

The m7100os is a software entry only, so there is no prototype. However the OS was tested in the iMCU7100EVB evaluation board running several test programs to demonstrate its capabilities. The evaluation board has the interfaces for the serial port, Ethernet port, LEDs and other hardware that was used to test the system. Below is a photo of the development board, with a single debugging button installed on nINT2 pin, to invoke the kernel monitor/debugger included.

The photo below is the result of the DHCP test code once the IP address is assigned. The debug button is on the bottom left.

Several examples of the use of the m7100os are included in the samples directory of the source code. The samples are accompanied by a video in the video subdirectory that shows the m7100os working. The most complete sample is the SNTP demo, which creates two tasks, one that synchronizes the internal software RTC with an NTP server and the other task reports the time via a telnet connection.

Code Sample

Two segments of code are shown in this section. The first one is how a user code is programmed to use the m7100os. The second one is a part of the Simple Network Time Protocol (SNTP) module demonstrating the simplicity of the socket interface and some other functions.

The following code shows a m7100os program with a task that sends pings to a server:

```
// some code before this deleted for space, please refer to the main.c ping sample code

void main() {
	uint8 i;

	// initialize system
	sys_initialize();

	// lower priority
	sys_set_priority(PRIORITY_NORMAL);

	// initialize lcd
	lcd_init();

	// initializing
	serial_puts("\n\r\n\rinitializing...\n\r");

	// print some chars
	lcd_print(0, "DC Monitor");

	// initialize network
	lcd_print(1, "TCP/IP init");
	sys_init_mac_address(mac);
	sys_init_ip(ip, nm, gw);
	set_MEMsize(txsize, rxsize);

	// wait until link
	// the iMCU7100EVB has the nLINK line connected to nINT3
	while (!(EIF & 0x02));
	EIF &= ~0x02;

#if OPTION_DHCP
	lcd_print(1, "DHCP request ");
	// DHCP request
	dhcp_client();
#endif

	// update ip array
	for (i = 0; i < 4; ++i)
		ip[i] = IINCHIP_READ(SIPR0+i);

	// print IP address in the LCD
	pu_sip(str, ip, STR_LEN);
	lcd_print(1, str);

	// create ping task
	sys_create_task(2, sysmon, sysmon_stack, MAX_STACK_SIZE, PRIORITY_NORMAL);
	EVB_nLED2 = 0;

} // main
```

```
#define BUF_SIZE    1024
__xdata uint8 buffer[BUF_SIZE];

uint8 fromip[4], toip[4];
uint16 fromport;
uint8 c[32];

void sysmon()
{
      uint8 t = 0;

      EVB_nLED0 = 0;

      // send ping
      toip[0] = 172;
      toip[1] = 16;
      toip[2] = 2;
      toip[3] = 3;

      for(;;) {
            // wait 2 seconds
            sys_sleep(TO_TICKS(2000000));

            serial_puts("sending ping to "); pu_ip(toip); serial_puts("\n\r");
            if (ping(toip, fromip, 1, TO_TICKS(1000000))) {
                  serial_puts("PING OK from "); pu_ip(fromip); serial_puts("\n\r");
            }
            EVB_nLED1 = t;
            t = 1 - t;
      }
}
```

This code is part of the SNTP module (sntp.c) demonstrating the simplicity of the POSIX alike socket interface and some other functions:

```
// sntp query
//   send a SNTP packet to the NTP server, get the time and set the parameter
//   return 0 if error
uint8 sntp(uint32 *rtc, uint16 *prescaler) __reentrant
{
      int8 s;
      __xdata struct sntp_packet *d = (__xdata struct sntp_packet *)sntp_buffer;
      uint8 i;
      __xdata uint8 *p;
      int16 l;
      unsigned long ts;
      uint32 t1, t2, ref;

      // check initialization
      if (!sntp_buffer_len || !sntp_buffer || !rtc) return 0;

      // create socket for query
      s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
      if (s < 0) return 0;
      setsockopt(s, N_DELAY, TO_TICKS(1000000));
      bind(s, NTP_PORT);
```

```
	// build SNTP UDP request packet
	// clear NTP packet
	for (i = 0; i < sizeof(struct sntp_packet); ++i) sntp_buffer[i] = 0;
	p = (uint8 *)sntp_buffer;
	// set control 11:011:011 00000000 00000100 11111010
	*p++ = 0xdb; *p++ = 0x00; *p++ = 0x04; *p++ = 0xfa;
	l = sizeof(struct sntp_packet);
	d->rootdelay  = reorder(0x00010000);
	d->dispersion = reorder(0x00010000);

	// send SNTP query packet to NTP server
	sendto(s, sntp_buffer, l, ntp_ip, NTP_PORT);

	// wait for answer
	l = 0;
	for (ts = get_systick() + TO_TICKS(SNTP_RESPONSE_DELAY); l <= 0 && get_systick() <
ts;) {
		l = recvfrom(s, sntp_buffer, sntp_buffer_len, NULL, 0);
	}
	ref = get_systick();
	// close socket
	close(s);

	if (l <= 0) return 0;

	// get time
	__critical {
		t1 = reorder(d->trans_ts[0]);                  // get transmit time
		t1 -= UNIX_DIF;                                      // adjust to UNIX format
		t2 = reorder(d->trans_ts[1]) >> 16;            // t2 in 1/65536 seconds
		t2 += (get_systick() - ref) * 13;              // adjust compute time
		t2 = (t2*TIMEBASE_1HZ_TICKS+0x8000) >> 16;     // get prescaler value

		*rtc = t1;
		if (prescaler) *prescaler = t2;
	}

	return 1;

} /* sntp */
```