RENESAS Everywhere you imagine.
M16C PLATFORM
DESIGN CONTEST 2005

PORTABLE WEBSITE

By Lindsay Meek

ABSTRACT

This project describes the use of a M32C/84 as an embedded internet appliance, capable of hosting a website over a standard dialup connection. When this device connects itself to the internet, it announces its allocated IP address to a dynamic DNS service, allowing a fixed domain name to point directly to HTTP and FTP servers. This provides a simple, low-power web-hosting option for web sites.

The data served by the website is stored on a low-cost compact flash card, supporting complex websites which can be updated easily using the inbuilt FTP server, or externally using a PC with a compact flash reader. The M32C/84 approach used a minimum number of ICs, and allowed a relatively fast memory-mapped interface to the compact flash card.

1. SYSTEM BLOCK DIAGRAM

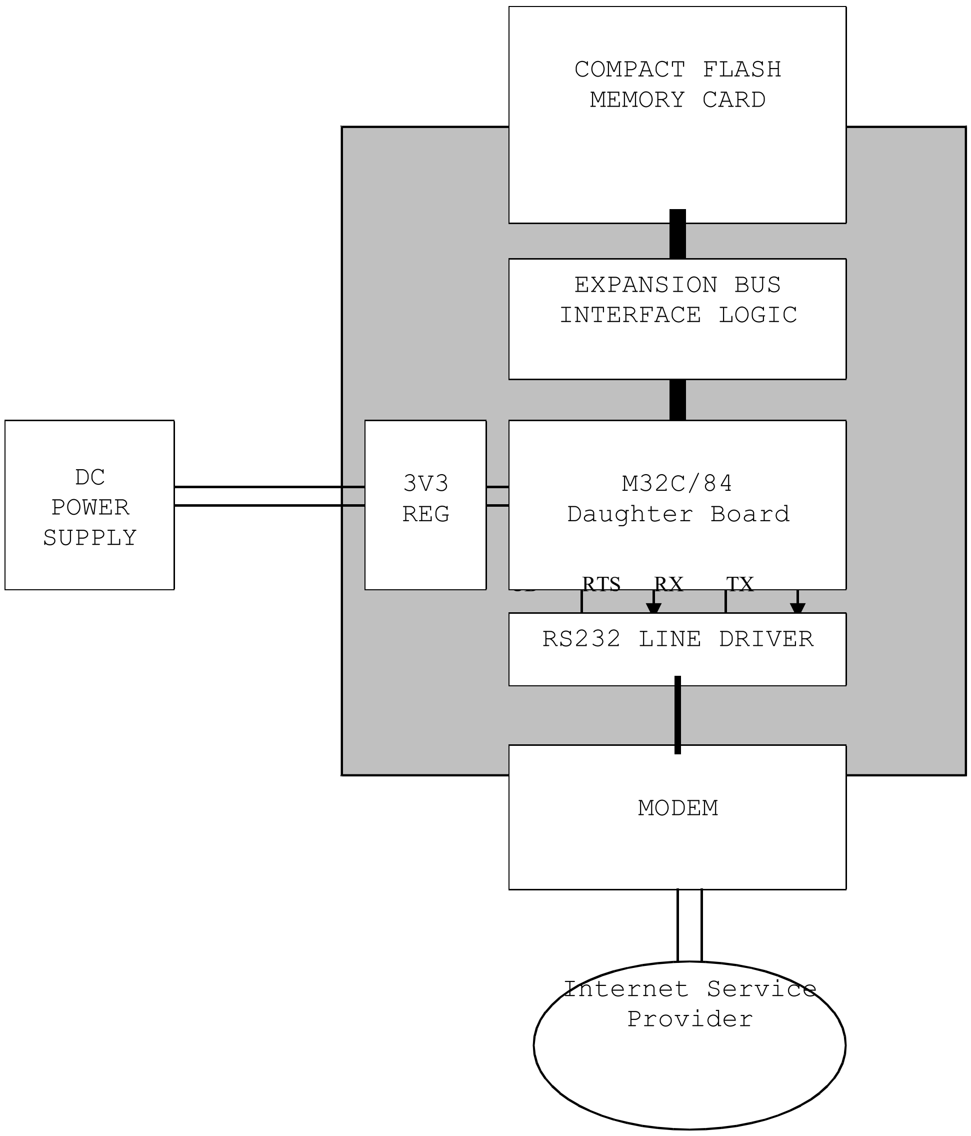
COMPACT FLASH MEMORY CARD
EXPANSION BUS INTERFACE LOGIC
DC POWER SUPPLY
3V3 REG
M32C/84 Daughter Board
RTS
RX
TX
RS232 LINE DRIVER
MODEM
Internet Service Provider

Figure 1. Website Module System Block Diagram

2. HARDWARE DESIGN

As indicated by the system block diagram, the main system components are the M32C/84 daughter board, a compact flash card interface, a RS232 modem interface and a power supply.

The M32C/84 forms the core of the system, and communicates directly with the compact flash card and modem using its memory expansion bus and UART interfaces.

2.1 COMPACT FLASH CARD INTERFACE

The compact flash card interface is used to connect compact flash memory cards, allowing the module access up to 4GB of low-cost solid-state memory memory.

The compact flash (CF) card interface is a minaturised version of the 16-bit PCMCIA connector that is commonly found on PDAs and laptops. It consists of a 16-bit data bus, with an 11-bit address space split into three main memory spaces; the I/O memory space, the common memory/attribute space. The common memory space is further decoded as attribute memory if the /REG line is asserted. Attribute memory contains the plug-and-play configuration-specific data for the card.

This was interfaced to the M32C expansion memory bus in non-multiplexed mode 3, using a single chip select line /CS0 combined with M32C address lines A10 and A11 to decode the various address spaces. This memory mapped the CF card in the memory area of E00000 hex, allowing rapid communications between the card and the CPU using memory access instructions or sout/sin style block-copy instructions.

The A10 line was directly connected to the CF /REG address line, and A11 was used to decode the I/O or common memory. Note that A10 is also present on the CF connector, however compact flash cards only use the first four address lines. In addition, the /BHE and A0 lines were used to enable the 8-bit byte lanes allowing 8 or 16-bit access. The signal mappings are probably best described by the following VHDL statements:

```
--
-- generate chip selects for 8/16 bit operation
--
chip_selects: process(cs0, bhe, a0)
begin
      ce0 <= '1';
      ce1 <= '1';
      if cs0='0' and a0='0' then
            ce0 <= '0';
      end if;
      if cs0='0' and bhe='0' then
            ce1 <= '0';
      end if;
end process chip_selects;

--
-- generate read and write strobes
--
read_strobes: process(rd, a11)
```

```
begin
      oe <= '1';
      iord <= '1';
      if rd='0' and a11='0' then
            oe <= '0';
      end if;
      if rd='0' and a11='1' then
            iord <= '0';
      end if;
end process read_strobes;

write_strobes: process(wr, a11)
begin
      we <= '1';
      iowr <= '1';
      if wr='0' and a11='0' then
            we <= '0';
      end if;
      if wr='0' and a11='1' then
            iowr <= '0';
      end if;
end process write_strobes;
```

Note that the CE0, CE1, IORD, IOWR, OE, and WE signals are CF bus signals. The /WAIT wait-state input to the M32C was directly connected to the CF RDY signal.

A Xilinx 9572XL PLD was selected to perform the signal mapping, as this part is readily available and could easily be altered if required after the PCB had been constructed. This could probably be simplified on a future PCB revision using discrete logic gates.

The memory expansion timing was configured to provide the minimum 600ns cycle time as required by the compact flash specification for attribute memory timing.

In addition, a dedicated PIO line was used to provide the RESET line to the CF card. This was necessary so the software driver could reset the card to a known state during the initialisation and configuration process.

2.2 RS232 INTERFACE

The RS232 interface was used to connect an external modem to the website module. The interface circuitry consisted of a generic charge-pump voltage converter, providing RX, TX, RTS and CD. The RTS line was used to provide hardware flow control to stop the modem from overflowing the input buffer during high-speed transmission bursts. The M32C/84 UART0 peripheral was used to communicate with the modem.

2.3 POWER SUPPLY

The power supply for the website module consisted of a 3.3V regulated DC supply, which was provided using a linear regulator. The compact flash cards support either 5V or 3.3V during their configuration phase, but most of the newer cards are designed to run at 3.3V.

The 3.3V power budget for the module was estimated to be

CF card	30 mA
M32C/84	22 mA
LCD Module	1 mA
LEDs (4)	14 mA
RS232 Converter	2 mA
9572XL-7	16 mA
TOTAL	85 mA

Regulator Power Dissipation (5-3.3)*85 = 144 mW
Regulator Temperature Rise (No Heatsink) = ~9 C

This indicates the linear regulator could easily manage with a 5V supply.

The reduced supply voltage required the M32C/84 sample board to be modified slightly, as the LCD module was intended to run at 5V. This required the contrast pot and logic supply input to be rewired so they were supplied directly from the pre-regulated (5V) voltage.

The PLL oscillator frequency of the M32C/84 was also slowed down from 32 MHz to 24 MHz to remain within specifications.

2.4 CIRCUIT SCHEMATIC

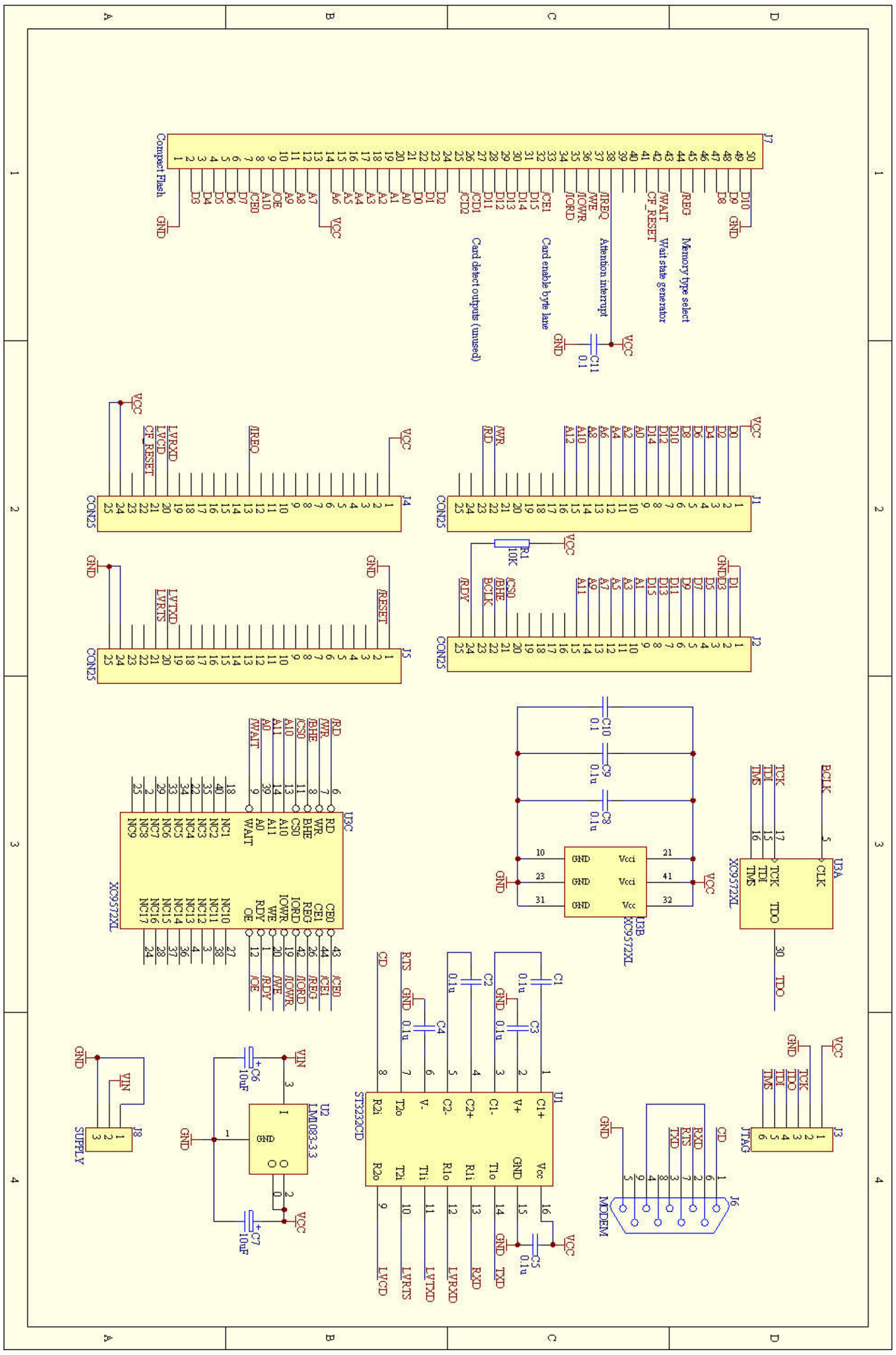

Figure 2. Website Circuit Schematic

3. SOFTWARE DESIGN

The main software elements of the website module can be broadly divided into the compact flash file system and the TCP/IP protocol stack. In order to accelerate the development cycle, the basic architecture was built with open-source software, and connected together with additional code to provide the desired functionality.

The main control flow of the web module can be summarised as

```
INITIALISE PLL & MCU CONFIGURATION
INITIALISE EXPANSION BUS
INITIALISE LCD DISPLAY
INITIALISE TICK TIMER
INITIALISE LEDS & SWITCHES
DISPLAY SIGN ON MESSAGE
INITIALISE UART
RESET & CONFIGURE COMPACT FLASH CARD FOR I/O MODE OPERATION
LOAD CONFIGURATION SETTINGS FROM FLASH CARD
DIAL ISP IF ENABLED
NEGOTIATE PPP CONNECTION
INITIALISE TCP PROTOCOL STACK
INSTALL HTTP AND FTP SERVERS ON PORTS 80 AND 21
INITIALISE DYNAMIC DNS CLIENT
FOR EVER
      SERVE FILE REQUESTS ON HTTP PORT
      SERVE FILE REQUESTS ON FTP PORT
      PERIODICALLY REFRESH DYNAMIC DNS SERVER IP ADDRESS
      IF DIAL ENABLED AND CARRIER NOT DETECTED
            RESET CPU
      END IF
LOOP
```

3.1 COMPACT FLASH CARD DRIVER

The compact flash interface made use of the M32C expansion bus to communicate with the compact flash card, using direct memory accesses.

The lowest layer of the compact flash card driver consisted of the basic I/O operations to access the three memory spaces present on the compact flash card. These were described in detail previously in section 2.1.

The next layer up was concerned with resetting & configuring the compact flash card into the correct operating mode. This process consisted of examining the attribute memory to verify the card type and determine the location of the PCMCIA configuration register. Once this is located, the card was configured into I/O mode, allowing the disk control registers to be accessed using the I/O space.

The next layer above card configuration is the basic IDE/ATA drive interface. This provided the simple disk related functions of identify device, read sector, and write sector.

Finally, a FAT16 file system driver was used to interpret the disk sector structure and provide a POSIX compliant file system interface consisting of the universal open(), close(), read(), write().. functions.

3.2 PROTOCOL STACKS

The protocol stacks employed by the website module consisted of the industry standard PPP, TCP/IP Sockets, FTP and HTTP servers. The stacks were primarily based on the TinyTCP/FTP package, and this was adapted to interface to the PPP client and FAT file system driver. Finally, a basic HTTP file server was added, together with carrier-detect monitoring for sensing dropped calls.

The HTTP server had the limitation of only sending a single page/file at a time, however this shouldn't impact performance as the modem will most likely be the largest communication bottleneck.

3.3 DYNAMIC DNS CLIENT

One of the features of the website module is its ability to appear as a static web site on the internet, even though its IP address changes each time it connects to an ISP.

This is achieved using a Dynamic DNS service, such as www.dynu.com, which provides a simple automated interface for redirecting DNS queries. It operates by issuing free subdomains off dynu.com, for example renesas.dynu.com. It is then possible to instantaneously set the IP address that this resolves to by issuing a simple web client request to the dynu web server.

The dynamic dns client within the website module has a periodic timer, which creates a web client request every 5 minutes. This sends the local IP address to the dynu server, thus creating a "virtual" static website.

4. PROJECT PHOTO

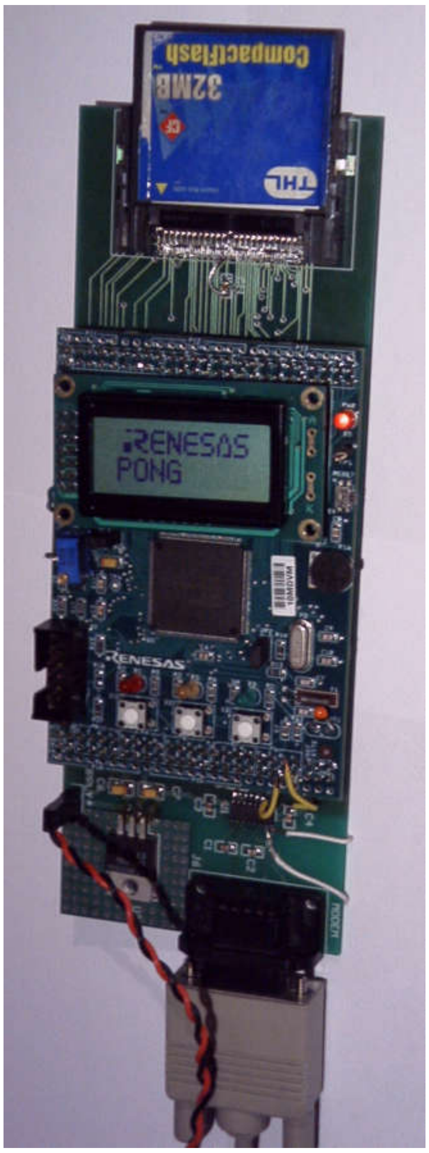

Figure 3. Project PCB with PING/PONG ICMP response shown

5. SYSTEM LIMITATIONS AND FUTURE WORK

During the course of the development, some observations were made on the limitations of the design, which could be addressed in a future revision.

The first limitation that can be addressed is the relatively inefficient implementation of stream and packet buffering, which resulted in a large RAM requirement for a relatively small number of socket handlers. A better approach would be to streamline and recycle the buffers, similar to the "socket memory" concept used by Linux. The improved efficiency would allow more than two simultaneous socket connections, and therefore greater data throughput.

The substitution of the carrier detect (CD) input with a clear to send (CTS) signal from the modem would also allow the module to make greater use of the buffers contained within the modem, and reduce the size of the local transmit stream buffer.

The FAT file system interface used on the module also does not currently support long file names (LFNs), which resulted in the need to truncate the website filenames to the 8.3 convention.

6. REFERENCES

CF+ & CompactFlash Specification Revision 3.0
http://www.compactflash.org/specdl1.htm

Anderson, D. PCMCIA System Architecture: 16-bit PC Cards, 2nd Ed. Addison Wesley
http://www.pcmcia.org/products/sa-pcm.htm

Compact Flash Memory Card Datasheet
http://www.sandisk.com/pdf/oem/cf-manual-10.8.pdf

Embedded Wi-Fi with TRENDnet
http://www.circuitcellar.com/library/print/0105/Eady174/

FAT16 File System Specification
http://www.maverick-os.dk/FileSystemFormats/FAT16_FileSystem.html

Apple II IDE/ATA interface
http://s.guillard.free.fr/Apple2IDE/

DYNU Dynamic DNS Service
http://www.dynu.com/basic.asp

TinyTCP
http://www.unusualresearch.com/tinytcp/tinytcp.htm

PPP Client Application Note 724
http://ww1.microchip.com/downloads/en/AppNotes/00724c.pdf

RFC 1661 - The Point-to-Point Protocol (PPP)
http://www.faqs.org/rfcs/rfc1661.html

RFC 791 - Internet Protocol
http://www.faqs.org/rfcs/rfc791.html

RFC 792 - Internet Control Message Protocol
http://www.faqs.org/rfcs/rfc792.html

RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0
http://www.faqs.org/rfcs/rfc1945.html

RFC 959 - File Transfer Protocol
http://www.faqs.org/rfcs/rfc959.html