

# Developing embedded devices using opensource tools: application to handheld game consoles

G. Goavec-Merou, S. Guinot, J.-M Friedt

Association Projet Aurore, Besançon, France

manuscript, slides and associated documents at  
<http://jmfriedt.free.fr>

August 14, 2009

# Introduction

- Hardware is hardly understandable in modern computers: serial (SATA, I<sup>2</sup>C) or fast (PCI) protocols difficult to understand
- Parallel protocols – easiest to use – are no longer available (ISA, processor bus 6502 or Z80, parallel port)
- Older, well documented hardware is still accessible on handheld gameconsoles.
- availability of an opensource emulator desmume

⇒ use the handheld game console Nintendo Dual Screen (NDS) for getting familiar with hardware/software interaction and instrument developement

This is **not** “yet another platform on which to run GNU/Linux”. The principles described here are valid for any platform: the NDS is widely available and “low” cost, well documented.

# Available hardware

- Two processors: ARM9 CPU and ARM7 coprocessor (bus sharing)  
⇒ ARM crosscompiler (gcc, newlib and binutils, appropriately patched depending on the target)
- 4 MB RAM (DS and DSLite: avoid DSi)
- wifi interface, but *no* asynchronous serial port (RS232)
- a legacy bus for Gameboy Advance compatibility: slot2
- a synchronous serial bus for reading game software: slot1 cartridge.

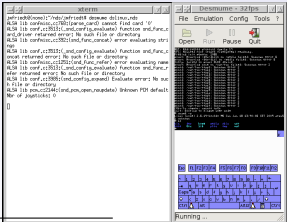
Requirement: get a cartridge for executing our own programs (games) on the NDS (M3DS Real).

**Objective:** display and transfer over the wifi network some “real world” data (sensor node, robotics ...)

# Let's start with a familiar environment

- Ready to use DSLinux image: Linux port to ARM9 processor thanks to the availability of gcc <sup>1</sup> (+newlib & binutils + patches)
- Familiar environment: posix, most hardware accessed through hardware modules /dev, no need to understand the underlying architecture, keyboard for typing commands on the bottom touchscreen
- Pre-compiled toolchain and linux image at <http://kineox.free.fr/DS/>

⇒ shell and “simple” interfaces such as framebuffer (/dev/fb0) are readily available thanks to the work of the DSLinux team



<sup>1</sup>toolchain compiled for x86 platform:  
<http://stsp.spline.de/dslinux/toolchain>

# Let's start with a familiar environment

Introductions

DSLlinux

Digital output  
Analog input

RTEMS

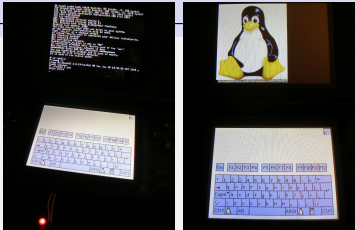
Draw, compute,  
RT  
wifi

PSP

uClinux &  
bootloader  
Serial  
communication

Conclusion

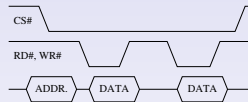
```
struct fb_var_screeninfo sinfo;  
unsigned short * s_ptr;  
  
inline void draw_pixel(int x, int y, int color)  
{ unsigned short *loc = s_ptr + \  
  ((y+sinfo.yoffset)*sinfo.xres)+x+sinfo.xoffset;  
  *loc = color; // 5R, 5G, 5B  
  *loc |= 1 << 15; // transparency ?  
}  
  
int main(int argc, char *argv[])  
{ char c;  
  screen_fd = open("/dev/fb0", O_RDWR);  
  ioctl(screen_fd, FBIOGET_VSCREENINFO, &sinfo);  
  
  s_ptr = mmap(0, screen_height*screen_width/8, PROT_READ|PROT_WRITE, MAP_SHARED, screen_fd, 0);  
  [...]  
}
```



- Understand the framebuffer data organisation (16 bit depth=5R5G5B)
- common interface with classical framebuffer: port from one architecture to another is mostly transparent *as long as an hardware abstraction layer exists*
- /dev/ and /proc are supported

## Slot2 cartridge bus

- All processors are based on the same architecture: a data bus holds the information (what), the address bus the location where the data are fetched/stored (where), the control signals indicate which operation to perform (read, write, interrupt, dma ...)
- The older gameboy advance – including ARM7 bus – was well documented

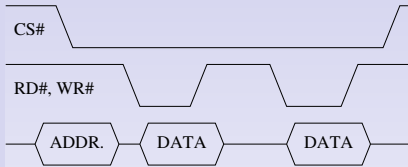


Let's try to use it to interface to the world ...

Two steps:

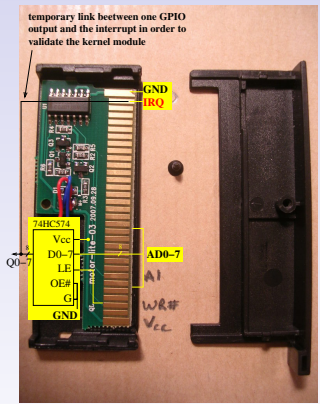
- 1 write informations to communicate with the world (no risk of damaging the hardware since output only)
- 2 read informations to gather informations on the environment (more challenging since the hardware must comply with the other peripherals of the CPU: but timing, high impedance bus)

# Hardware interfacing



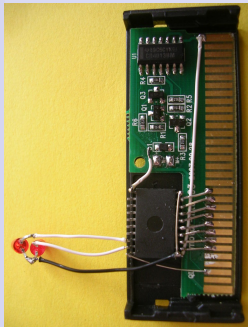
Parallel bus  $\Rightarrow$

- address defines *where* the action takes place,
- data bus defines *what* is transfered,
- control bus defines *how* the transfer occurs



# Blinking LED

- No need for fancy hardware: use a Rumble Pack cartridge
- Program example for accessing a hardware address (memory mapped I/O in Freescale architecture):



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

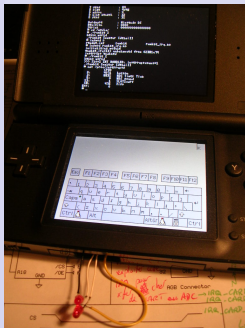
int main(int argc, char **argv)
{ printf("demo rumble : 1/3=%f\n", 1./3.);
  if (argc > 1) {
    *(unsigned short*)(0x8000000)=(unsigned short)atoi(argv[1]);
    sleep(1); // active le moteur sur argv →
              ↪ [1]=2
    *(unsigned short*)(0x8000000)=0;
  }
  return (0);
}
```

- 1 define which address is associated with which hardware (address decoder)
- 2 define the data size (\*(unsigned short\*))
- 3 define which value to put on the data bus
- 4 the control signal are automatically generated by the processor



# Blinking LED

- No need for fancy hardware: use a Rumble Pack cartridge
- Program example for accessing a hardware address (memory mapped I/O in Freescale architecture):



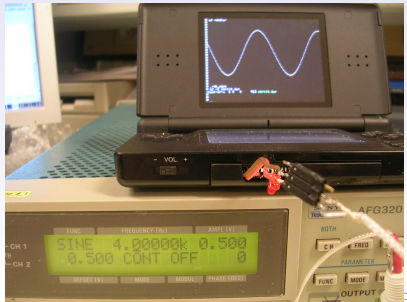
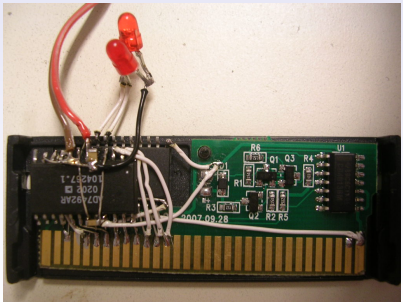
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
{ printf("demo rumble : 1/3=%f\n", 1./3.);
  if (argc>1) {
    *(unsigned short*)(0x8000000)=(unsigned short)atoi(argv[1]);
    sleep(1); // active le moteur sur argv
    ↪ [1]=2
    *(unsigned short*)(0x8000000)=0;
  }
  return (0);
}
```

- 1 define which address is associated with which hardware (address decoder)
- 2 define the data size (\*(unsigned short\*))
- 3 define which value to put on the data bus
- 4 the control signal are automatically generated by the processor

# Data acquisition

- The only additional trick is to *keep the data bus lines high impedance* when you are not supposed to talk
- Use of an analog to digital converter *in parallel* (bus sharing thanks to RD#/WR# and CS#) to the latch
- This example: fast analog to digital conversion (ADC), theoretically 1 MS/s, practically half that speed
- ADC is more fancy than a latch: start conversion, wait, read conversion result (fixed delay or interrupt = kernel module)



# ADC control example

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#define TAILLE 255

int main(int argc, char **argv)
{
    int f, taille=TAILLE;
    volatile int k;
    char *c;

    c=(char*)malloc(TAILLE); // demonstrates the use of malloc without MMU
    for (f=0;f<TAILLE;f++)
    {
        *(unsigned short*)(0x8000000)=(unsigned short)0;
        for (k=0;k<10;k++) {} // DO NOT compile with -O2
        // usleep(7); // call to usleep is too long !
        c[f]=*(unsigned short*)(0x8000000)&0xff;
    }
    for (f=0;f<TAILLE;f++) printf("%x ",c[f]); printf("\n");
    return(0);
}
```

- user-space memory mapped device with fixed delay (uClinux  $\Leftrightarrow$  no MMU)
- kernel space with fixed delay
- hardware interrupt generated by end-of-conversion (slow !)

# Problem with uClinux

Conclusion of DSLinux: **memory footprint too large** for a 4 MB system  
⇒ follow the trend and add RAM, or **find a better use of the available resources**.

- most programs in busybox will run out of memory, even less !  
only a few hundred kB remain after loading the kernel
- no luck in using portable graphic interfaces (SDL, Qtopia)
- one option is to use the memory expansion pack ... on slot 2, *i.e.* no access to the ARM9 bus left

Yet another solution: use another development environment, POSIX compatible yet with a small footprint.

# RTEMS on NDS

- RTEMS is an *executive environment* for embedded devices, *i.e.* a framework for developing a monolithic application based on multiple modules.
- From the developer point of view, similar to an operating system, but no memory management, no scheduler, no dynamic loading of application ...
- Yet availability of OS functionalities such as TCP/IP stack, file format, threads ... even a shell & user input interface (Graffiti) !
- Dedicated functionalities for embedded system debugging: CPU use, stack use
- Real time + strong support (used by large national & international agencies)
- compiled with gcc (+ patches)  $\Rightarrow$  familiar environment

**An NDS BSP has been developed by M. Bucchianeri, B. Ratier, R. Voltz & C. Gestes <sup>2</sup>**

<sup>2</sup>[http://www.rtems.com/ftp/pub/rtems/current\\_contrib/nds-bsp/manual.html](http://www.rtems.com/ftp/pub/rtems/current_contrib/nds-bsp/manual.html)



```
#include <bsp.h>
#include <stdlib.h>
#include <stdio.h>
#include <nds/memory.h>

rtems_id timer_id;
uint16_t l=0;

void callback()
{
    printf(" Callback %x\n",l);
    (*(volatile uint16_t*)0x08000000)=l;
    l=0xffff-l;
    rtems_timer_fire_after(timer_id, 100, callback, NULL);
}

rtems_task Init(rtems_task_argument ignored)
{
    rtems_status_code status;
    rtems_name timer_name = rtems_build_name('C','P','U','T');

    printf( "\n\n*** HELLO WORLD TEST ***\n" );
    (*(vuint16_t*)0x04000204) = ((*(vuint16_t*)0x04000204) & ~ARM7_OWNS_ROM); // bus access to ARM9

    status = rtems_timer_create(timer_name,&timer_id);
    rtems_timer_fire_after(timer_id, 1, callback, NULL);
    rtems_stack_checker_report_usage(); // requires #define CONFIGURE_INIT

    printf( "*** END OF HELLO WORLD TEST ***\n" );
    while(1) ;
    exit( 0 );
}

/* configuration information */
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

/* configuration information */
#define CONFIGURE_MAXIMUM_DEVICES          40
#define CONFIGURE_MAXIMUM_TASKS           100
#define CONFIGURE_MAXIMUM_TIMERS           32
#define CONFIGURE_MAXIMUM_SEMAPHORES      100
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES  20
#define CONFIGURE_MAXIMUM_PARTITIONS       100
#define CONFIGURE_MAXIMUM_REGIONS         100
```

## Framebuffer access & math lib.

Framebuffer access similar to DSLinux: minor changes to adapt to the new device naming convention (exemple provided by M. Bucchianeri)

```
inline void draw_pixel(int x, int y, int color)
{ uint16_t* loc = fb_info.smem_start;
  loc += y * fb_info.xres + x;
  *loc = color; *loc |= 1 << 15; // 5R, 5G, 5B
}

...
struct fb_exec_function exec;
int fd = open("/dev/fb0", O_RDWR);
exec.func_no = FB_FUNC_ENTER_GRAPHICS;
ioctl(fd, FB_EXEC_FUNCTION, (void*)&exec);
ioctl(fd, FB_SCREENINFO, (void*)&fb_info);
...
#define CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE

rtems_driver_address_table Device_drivers[] =
{ CONSOLE_DRIVER_TABLE_ENTRY,
  CLOCK_DRIVER_TABLE_ENTRY,
  FB_DRIVER_TABLE_ENTRY,
  { NULL,NULL, NULL,NULL,NULL, NULL }
};
```



Drawing of a fractal displaying the resolution of a 3rd order polynom:

- ① the NDS can be used for useful calculations including solving equations: floating point calculation emulation
- ② access to framebuffer
- ③ adding a new command to the RTEMS shell



# Hardware access (GPIO & ADC)

Bus control must be granted to the ARM9 CPU

```
#include <nds/memory.h>
```

```
[...]
```

```
void callback()
```

```
{ printk("Callback %x\n",l);
```

```
    (*(volatile uint16_t*)0x08000000)=1;
```

```
    l=0xffff-1;
```

```
    rtems_timer_fire_after(timer_id, 100, callback, NULL);
```

```
}
```

```
rtems_task Init(rtems_task_argument ignored)
```

```
{rtems_status_code status;
```

```
    rtems_name timer_name = rtems_build_name('C','P','U','T');
```

```
// cf rtems-4.9.1/c/src/lib/libbsp/arm/nds/libnds/source/arm9/rumble.c
```

```
// sysSetCartOwner(BUS_OWNER_ARM9);
```

```
// defini dans rtems-4.9.1/c/src/lib/libbsp/arm/nds/libnds/include/nds/memo
```

```
    (*(vuint16*)0x04000204) = ((*(vuint16*)0x04000204) & ~ARM7_OWNS_ROM);
```

```
    status = rtems_timer_create(timer_name,&timer_id);
```

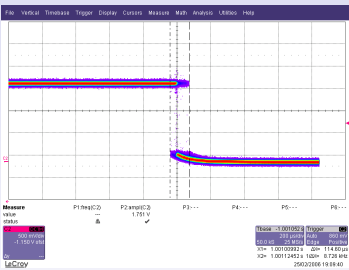
```
    rtems_timer_fire_after(timer_id, 1, callback, NULL);
```

```
[...]
```

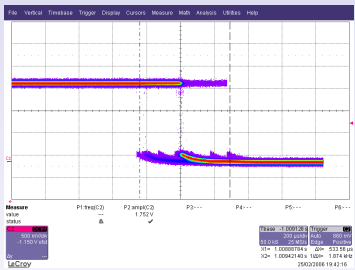
```
}
```

# Real time ?

- Real time is defined as a system with *maximum latency* between a signal and the processing of the information.
- The shorter the better, but an upper boundary must never be reached



RTEMS



DSLlinux

Three threads, two for blinking diodes and one running the Newton fractal drawing for  $\approx 10$  seconds upon user request.  
→ DSLinux displays latency shift during context *changes*.

# TCP/IP over wifi networking

## Introductions

## DSLlinux

Digital output  
Analog input

## RTEMS

Draw, compute,  
RT  
wifi

## PSP

uCLinux &  
bootloader

Serial  
communication

## Conclusion

- correction of a bug in the data structure handling TCP/IP packets
- convenient connection to the shell (as opposed to the Graffiti interface)
- use of the wireless interface for data transfer (the telnetd server can handle other applications than a shell)

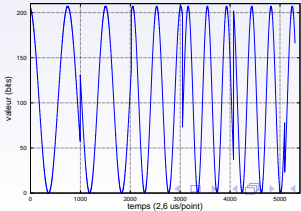
```
gwe@omniBook:~
Eterm Font Background Terminal
Destination Gateway/Host/Hub Flags Refs Use Expire Interface
default 10.0.1.1 UGS 0 0 0 dsifio
10.0.1.0 255.255.255.0 U 0 0 4 dsifio
10.0.1.1 0 0 4 dsifio
10.0.1.6 00:0D:60:CC:39:7E UHL 1 158 1209 dsifio
[/] # ifconfig
ifconfig command not found
[/] # ifconfig
ifconfig: ***** INTERFACE STATISTICS *****
***** dsifio *****
Ethernet Address: 00:1B:EA:A2:9B:EC
Address: 10.0.1.20 Broadcast Address: 10.0.1.255 Net mask: 255.255.255.0
Flags: Up Broadcast Running Active Simplex
Send queue limit: 150 length: 1 Dropped: 0
[/] # route
Destination Gateway/Host/Hub Flags Refs Use Expire Interface
default 10.0.1.1 UGS 0 0 0 dsifio
10.0.1.0 255.255.255.0 U 0 0 4 dsifio
10.0.1.1 0 0 4 dsifio
10.0.1.6 00:0D:60:CC:39:7E UHL 1 176 1209 dsifio
[/] # netstat
[/] # exit
Shell exiting
Welcome to RTEMS-4.2.1(ARM/ARHv4/rtd)
COPYRIGHT (c) 1989-2008,
On-Line Applications Research Corporation (OLAR),
Login into RTEMS
login: root
Password:
RTEMS SHELL (Ver.1.0-FRC)1, Apr 5 2009, 'help' to list commands.
[/] # ifconfig
ifconfig: ***** INTERFACE STATISTICS *****
***** dsifio *****
Ethernet Address: 00:1B:EA:A2:9B:EC
Address: 10.0.1.20 Broadcast Address: 10.0.1.255 Net mask: 255.255.255.0
Flags: Up Broadcast Running Active Simplex
Send queue limit: 150 length: 1 Dropped: 0
[/] # route
Destination Gateway/Host/Hub Flags Refs Use Expire Interface
default 10.0.1.1 UGS 0 0 0 dsifio
10.0.1.0 255.255.255.0 U 0 0 4 dsifio
10.0.1.1 0 0 4 dsifio
10.0.1.6 00:0D:60:CC:39:7E UHL 1 223 1209 dsifio
[/] #
```

Developing  
embedded  
devices using  
opensource tools:  
application to  
handheld game  
consoles

Friedt & al

# TCP/IP over wifi networking

- correction of a bug in the data structure handling TCP/IP packets
- convenient connection to the shell (as opposed to the Graffiti interface)
- use of the wireless interface for data transfer (the telnetd server can handle other applications than a shell)



# TCP/IP over wifi networking

## Introductions

### DSLinux

Digital output  
Analog input

### RTEMS

Draw, compute,  
RT  
wifi

### PSP

uClinux &  
bootloader

Serial  
communication

## Conclusion

## 1. ifconfig:

```
static struct rtems_bsdnet_ifconfig netdriver_config = {
    RTEMS_BSP_NETWORK_DRIVER_NAME,
    RTEMS_BSP_NETWORK_DRIVER_ATTACH,
    NULL,                /* No more interfaces */
    "10.0.1.20",          /* IP address */
    "255.255.255.0",      /* IP net mask */
    NULL,                /* Driver supplies hw addr */
};
```

## 2. route:

```
/* Network configuration */
struct rtems_bsdnet_config rtems_bsdnet_config = {
    &netdriver_config,
    NULL,                /* do not use bootp */
    0,                   /* Default network task priority */
    0,                   /* Default mbuf capacity */
    0,                   /* Default mbuf cluster capacity */
    "rtems",             /* Host name */
    "trabucayre.com",    /* Domain name */
    "10.0.1.1",          /* Gateway */
    "10.0.1.13",         /* Log host */
    {"10.0.1.13"},       /* Name server(s) */
    {"10.0.1.13"},       /* NTP server(s) */
};
```

## 3. telnetd: (server)

```
rtems_telnetd_initialize(
    rtemsShell,          /* "shell" function */
    NULL,               /* no context necessary for echoShell */
    false,              /* listen on sockets */
    RTEMS_MINIMUM_STACK_SIZE*20, /* shell needs a large stack */
    1,                  /* priority */
    false               /* telnetd does NOT ask for password */
);
```

## 4. application:

```
void rtemsShell(char *pty_name, void *cmd_arg) {
    printk("===== Starting Shell =====\n");
    rtems_shell_main_loop( NULL );
    printk("===== Exiting Shell =====\n");
}
```

## OR (another server than the shell)

```
void telnetADC( char *pty_name, void *cmd_arg) {
    char *c; int f;
    c=(char*)malloc(TAILLE);
    while (1) {
        for (f=0;f<TAILLE;f++) {
            *(unsigned short*)(0x80000000)=(unsigned short)0;
            c[f]**(unsigned short*)(0x80000000)&0xff;
        }
        for (f=0;f<TAILLE;f++) printf("%x ",c[f]);
        printf("\n");
    }
}
```

Developing  
embedded  
devices using  
opensource tools:  
application to  
handheld game  
consoles

Friedt & al

Introductions

DSLinux

Digital output  
Analog input

RTEMS

Draw, compute,  
RT  
wifi

PSP

uClinux &  
bootloader  
Serial  
communication

Conclusion

# TCP/IP over wifi networking



Configuration of AP MAC address (commercial game)



Direct connection from NDS to eeePC configured as AP <sup>3</sup>

⇒ fully functional TCP/IP stack for connection to the NDS running RTEMS through a wifi link

<sup>3</sup><http://jmfriedt.free.fr/fred/fred.html>

## uClinux on PSP

- So far we have used readily available toolchains (DSLinux, RTEMS) targetted towards ARM architectures.
- The toolset gcc, binutils & newlib will run on most architectures, including MIPS.
- The PSP is based on a MIPS4 architecture, provides 32 MB RAM
- Let's try to build a cross compilation environment for uClinux towards the PSP

Less hardware oriented and more towards software: the newer PSP has hardly any communication peripherals (RS232 as IR and headset port, USB & wifi not supported)  
⇒ external microcontroller for hardware interface and communicate through RS232



# Requirements

- Flash a new firmware to replace the original Sony firmware which does not allow executing programs from the MemoryStick.
- Install the crosscompilation toolchain (PSP SDK <sup>4</sup> and the toolset called the PSP SDK <sup>5</sup> to generate homebrew games: the bootloader copying the uClinux kernel + rootfs to RAM is a game
- The games have access to the Sony OS functions for accessing hardware peripherals. Most functions are *not* reverse engineered (IPL SDK) and hence hardware peripherals are not accessible from uClinux

<sup>4</sup><http://ps2dev.org/psp/Tools/Toolchain/psptoolchain-20070626.tar.bz2>

<sup>5</sup><http://ps2dev.org/psp/Projects/PSPSDK>



# The bootloader

Uncompress the kernel + rootfs image, copy to memory and jump to the starting address after initializing the hardware: last chance to use the Sony OS functions (e.g. `pspDebugScreenPrintf()`)

```
#define KERNEL_ENTRY          0x88000000
#define KERNEL_PARAM_OFFSET   0x00000008
#define KERNEL_MAX_SIZE       (size_t)( 4 * 1024 * 1024 )   /* 4M */

#define printf                pspDebugScreenPrintf

BOOL loadKernel(void ** buf_, int * size_)
{
    gzFile zf;
    void * buf;
    int size;

    zf = gzopen( s_paramKernel, "r" );
    buf = (void *)malloc( KERNEL_MAX_SIZE );
    size = gzread( zf, buf, KERNEL_MAX_SIZE );
    gzclose( zf );
    *buf_ = buf;
    *size_ = size;
}
/*-----*/
void transferControl(void * buf_, int size_)
{
    KernelEntryFunc kernelEntry = (KernelEntryFunc)( KERNEL_ENTRY );
    memcpy( (void *) ( KERNEL_ENTRY ), buf_, size_ ); /* prepare kernel image */
    uart3_setbaud( s_paramBaud );
    uart3_puts( "Booting Linux kernel...\n" );
    kernelEntry( 0, 0, kernelParam );
}
```

## uClinux for MIPS: toolchain

- The MIPS cross-compilation toolchain is known to work well: architecture used on routers
- MMU-less CPU on PSP  $\Rightarrow$  generate BFLT outputs instead of the usual ELF.
- requires a dedicated linker: `elf2flt` includes
  - the shell script `ld-elf2flt`
  - the program `elf2flt` able to convert ELF binaries to the Binary Flat (BFLT) format
  - the program `flthdr` for editing the header of BFLT files.
  - `ld-elf2flt` replaces the usual `ld` linker when the `-elf2flt` compiler option is used

If the `ld` script is called with the `-elf2flt` option, then `ld.real` (original linker) followed by `ld-elf2flt` are successively called.

```
# mipsel-psp-gcc -Wl,-elf2flt -o test test.c
# ls
test.gdb test
```

# uClinux for MIPS: buildroot

## Introductions

### DSLinux

Digital output  
Analog input

### RTEMS

Draw, compute,  
RT  
wifi

### PSP

uClinux &  
bootloader  
Serial  
communication

## Conclusion

- Linux kernel + associated tools for generating a roots = complex environment
- One of the most commonly used BSP which manages patches & Makefiles for generating a coherent environment for compiling the toolchain, kernel and userspace tools: buidroot
- buildroot = a structured set of Makefiles and patches for most CPU architectures + scripts for downloading the right archives to build a functional uClinux image

Result of the compilation: the cpio image including kernel + rootfs (including busybox) as initramfs at

`buildroot-psp/project.build_mipsel/linuxonpsp/linux-2.6.22/arch/mips/boot/vmlinux.bin.`

# Running uClinux and communication

uClinux runs on the PSP, the OnScreen Keyboard has been ported (SMS like)



- The PSP provides enough memory to run graphic interfaces such as SDL, so porting many GNU/Linux tools to the MIPS based PSP is possible
- what about hardware interfacing ?

Developing  
embedded  
devices using  
opensource tools:  
application to  
handheld game  
consoles

Friedt & al

Introductions

DSLinux

Digital output  
Analog input

RTMS

Draw, compute,  
RT  
wifi

PSP

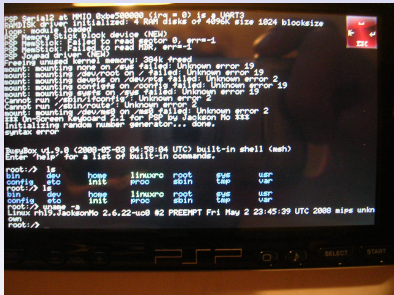
uClinux &  
bootloader

Serial  
communication

Conclusion

# Running uClinux and communication

uClinux runs on the PSP, the OnScreen Keyboard has been ported (SMS like)



- The PSP provides enough memory to run graphic interfaces such as SDL, so porting many GNU/Linux tools to the MIPS based PSP is possible
- what about hardware interfacing ?

# Using the serial port to communicate with the PSP

- First approach: use a terminal software since the serial port has been linked to the keyboard layer
  - the serial port is periodically polled (CPUTIMER interrupt 66), cf `linux/drivers/serial/serial_psp.c`
  - `void psp_uart3_txx_tick()`

```
{  
    if ( !s_psp_uart3_port_data.shutdown &&  
        ( s_psp_uart3_port_data.txStarted ||  
          ( !s_psp_uart3_port_data.rxStopped &&  
            !( PSP_UART3_STATUS & PSP_UART3_MASK_RXEMPTY ) ) ) )  
    {        // wakes up the kernel thread for receiving char  
        up( &s_psp_uart3_port_data.sem );  
    }  
}
```
  - this interrupt service routine wakes up a kernel thread in charge of testing whether a character is available in the queue of the UART
- use the serial port and a dedicated microcontroller to add hardware functionalities

# Converting a PS2 keyboard stream to RS232



- Use of one of the examples provided with `msp-gcc` (the `gcc` cross-compiler targeting the MSP430) in `examples/mspgcc/pc_keyboard`.
- instead of displaying the keystroke on an LCD, transfer via RS232
- notice that the key number is transferred on PS2 when the key is hit *and released*
- no need for a PC, autonomous PSP with keyboard !

## Conclusion

- A game console provides the resources typically found in high grade embedded systems (e.g. routers, digital cameras) and hence a playground to get familiar the techniques associated with scarce resource
- NDS hardware bus still understandable and usable for hardware interfacing
- gcc is our friend, with a ports to MIPS and ARM-architectures
- DSLinux requires too much memory to perform any useful function
- switch to a low memory footprint executive environment: RTEMS
- patched RTEMS for NDS to add full wifi communication functionality  $\Rightarrow$  initial goal reached, *i.e.* wireless transmission of physical quantities obtained on an A/D converter
- coherent environment to compile uClinux + tools on MIPS based PSP
- little hardware extension capabilities on PSP, so add external microcontroler communicating through RS232 link
- use and expand some of the available demonstration applications to suite most of our needs (framebuffer, text mode interface, character input ...)



# Acknowledgement

- Pierre Kestener (CEA/IRFU, Saclay, France) mentioned the NDS BSP of RTEMS
- M. Bucchianeri answered our questions concerning the use of the RTEMS BSP
- Santa Claus brought the PSP and NDS handheld game consoles

## Further readings:

- S. Guinot & J.-M. Friedt, *GNU/Linux sur Playstation Portable*, GNU/Linux Magazine France 114, March 2009, pp.30-40 [in French]
- J.-M Friedt & G. Goavec-Merou, *Interfaces matérielles et OS libres pour Nintendo DS : DSLinux et RTEMS*, GNU/Linux Magazine France **Hors Série 43** (August 2009) [in French] (and included references)
- these articles were partially translated for HAR2009, the resulting article is available at  
<https://har2009.org/program/events/37.en.html>