Introduction to uClinux: low cost embedded-device development environments

J.-M Friedt, S. Guinot
Association Projet Aurore (Besançon, France)

slides available at `http://jmfriedt.free.fr`

airborne pictures available at
`http://projetaurore.assos.univ-fcomte.fr`

software archive available at `http://www.sequanux.org`

July 5, 2005

What is uClinux: linux for embedded devices

- processor without MMU
- low memory footprint
- low power consumption processors (Motorola/Freescale Coldfire)

⇒ Intermediate between microcontrollers and single board computers
⇒ Solution cheaper than PC104/biscuit PCs

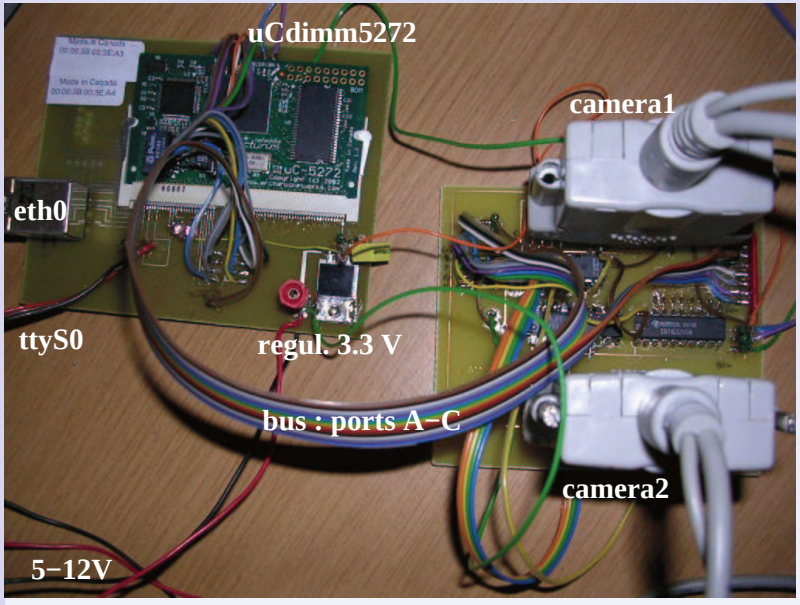
UNIVERSITÉ DE FRANCHE-COMTÉ
PROJET AURORE
DIFFUSION DE LA CULTURE ET DE L'INFORMATION SCIENTIFIQUE ET TECHNIQUE

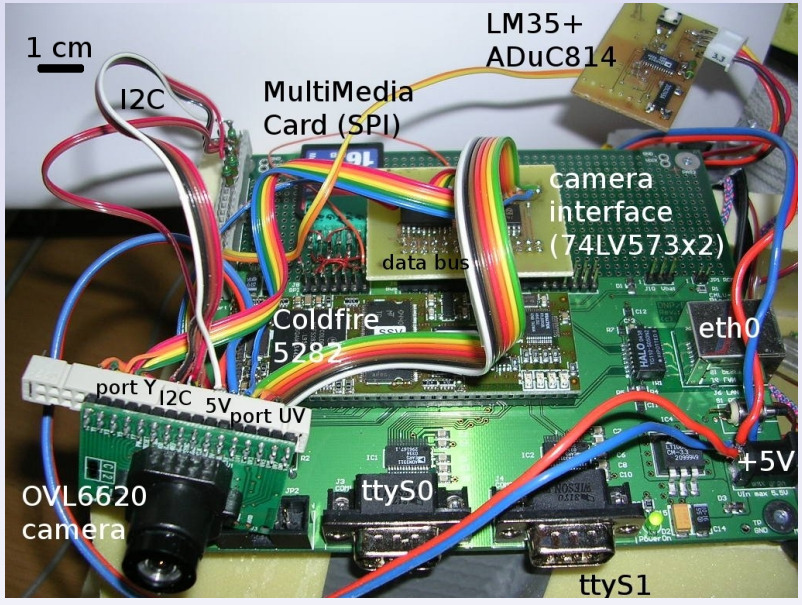
sequanux
Logiciels libres en Franche-Comté

Hardware aspects

Hardware used in this presentation:

- Coldfire 5272 (Arcturus Networks, Canada), SODIMM144 connector
- Coldfire 5282 (SSV, Germany), DIL64 connector





Similarities:

- 2 RS232, SPI, 10 Mb ethernet, 100 Mb ethernet,
- several MB flash/RAM,
- fast processors (66 MHz)

Differences: ADC/USB, I^2C/PWM

Objectives

Our final objective:
develop tools for remote sensing, autonomous robotics/vehicles, instrumentation [1]
Physics is a driving force of computer development: cf web [2]

[1] J.-M Friedt, S. Guinot, É. Carry, *Introduction au Coldfire 5272*, GNU/Linux Magazine France **73** (June 2005) pp.26-33 [in French] – English translation available at `http://jmfriedt.free.fr/uclinux_eng.pdf`

[2] J. Gillies, R. Caillau, *How the web was born*, Oxford Univ. Press (2000)

Why use uClinux ?

- scheduler (single user/multiple processes)
- memory management (malloc) $\Rightarrow$ limited contiguous memory allocation can be **overcome manually**
- networking (TCP/IP stack)
- libraries (pthread, jpeg, compression ...)
- uClibc: C library compatible with uClinux, static libraries only

Development environment

- cross compilation on a linux-running PC (generate Motorola binaries on an Intel CPU)
- mount PC partition by NFS
- run program on Coldfire and debug using the real hardware
- use the high bandwidth of ethernet to store results on PC for further analysis (`gnuplot`, graphics display ...)

```
# uname -a
uClinux pa5272 2.4.27-uc1 #15 lun mai 30 22:44:15 CEST 2005 m68knommu unknown
# ls /bin
traceroute  sh          route       umount      more        cp
camedia     portmap     basename    touch       mknod       chmod
shutdown    login       du          rmdir       mkdir       busybox
reboot      init        rmmod       rm          ls          agetty
qcam        inetd       lsmod       pwd         kill
telnetd     sync        insmod      ps          hostname
telnet      ln          ifconfig    ping        gzip
printenv    killall     devfsd      mv          dmesg
logname     free        uname       mount       df
```

uClinux/linux similarities

Advantage: linux programs are mostly compatible with uClinux

Example: Hello World with fpu emulation

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {printf("sqrt(%d)=%f\n",3,sqrt(3));return(0);}
```

Example: RS232 port access

```
int main(int argc,char **argv)
{int fd; extern struct termios oldtio,newtio;
 fd=open("/dev/ttyS1", O_RDWR | O_NOCTTY);
 tcgetattr(fd,&oldtio);
 newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; newtio.c_iflag = IGNPAR; newtio.c_oflag = ONOCR;
 newtio.c_cc[VTIME] = 0; newtio.c_cc[VMIN] = 1;
 tcsetattr(fd,TCSANOW,&newtio);
 while (1) {read(fd,buf,2);
            temp=(unsigned short)((buf[0]&0xff)*256+(buf[1]&0xff));
    printf ("%x=%.2f degC\n",temp,(float)temp/40.96*2.5);
    fflush(stdout);
   }
 return(0);}
```

D. McCullough, uClinux for Linux Programmers, http://www.linuxjournal.com/node/7221/print

uClinux/linux similarities

Kernel programming:
kernel modules are similar under linux and uClinux ⇒ faster to get used to than to a new OS (ECOS ? TinyOS ?)
Example: interrupt management on the 5272

```
#define IRQ_DEFAULT 66
static int uc_int_major = 6;

static struct file_operations uc_int_fops =
{owner: THIS_MODULE,
 read: uc_int_read,
 open: uc_int_open,
 release: uc_int_release,
};

static void uc_int_irqhandler (int irq, void *dev_id, struct pt_regs *regs)
{unsigned long tmp;
 [...]
 tmp = *((volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1));

 /* disable interrupts */
 tmp &=0xf8ffffff;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 /* we ack this interruption to the hardware */
 tmp |=0x08000000;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 /* and wake up user application waiting for a read */
 wake_up_interruptible (&dev->uc_int_queue);
 /* restore interrupts */
 tmp |= 0x0f000000;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 [...]
}
```

uClinux/linux similarities (2)

```
}
static int uc_int_open (struct inode *inode, struct file *file)
{[...]
 tmp = *((volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1));
 tmp |= 0x0f000000;
 *(volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1) = tmp;
 if ((retval = request_irq (uc_int_irq, uc_int_irqhandler, SA_INTERRUPT, "uc_int", dev)))
    {dbg("unable to assign irq %d", uc_int_irq);goto exit_sem;}
 [...]
}

static ssize_t uc_int_read (struct file *file, char *buffer, size_t count, loff_t *ppos)
{unsigned long tmp[4];
 [...]
 interruptible_sleep_on (&dev->uc_int_queue); /* sleep until a interrupt occurs */
 memcpy ((void *) tmp, (void *) (MCF_MBAR + MCFSIM_ICR1), sizeof (tmp));
 [...]
}

/* the realease method is call when user want close () the device */
static int uc_int_release (struct inode *inode, struct file *file)
{[...]
 free_irq (dev->uc_int_irq, NULL);
 [...]
}
```

uClinux/linux differences

Example: digital input/output (blinking LED on the 5272)

```
*((volatile unsigned char *)(MCF_MBAR+MCFSIM_PADAT))|=0x20;              // write PC
printf("%.4x ",*((volatile unsigned short*)(MCF_MBAR + MCFSIM_PCDAT ))); // read PA, PB
```

Here

- `MCFSIM_PCDAT` is the port register to be written to/read
- `MCF_MBAR` is a standard hardware register base address
- confusing difference between Arcturus uCdimm5272 and Freescale naming conventions

Using uClinux does not prevent (yet) from reading the datasheet of the processor

uClinux/linux differences

Consequence from the lack of MMU:

- no VM (virtual memory)
- no segmentation or pagination fault
- no fork sytem call. can't implement the "copy on write" strategy. An alternative comes with the old BSD `vfork` and all his limits $\Rightarrow$ porting difficulties
- no dynamic heap. Instead uclinux use a global memory pool (kernel free memory pool)

Large memory allocation under uClinux

No obvious implementation of `malloc` due to lack of heap

$\longrightarrow$ pick in available memory

$\Rightarrow$ fast

$\Rightarrow$ one process might request all available memory (no limit)

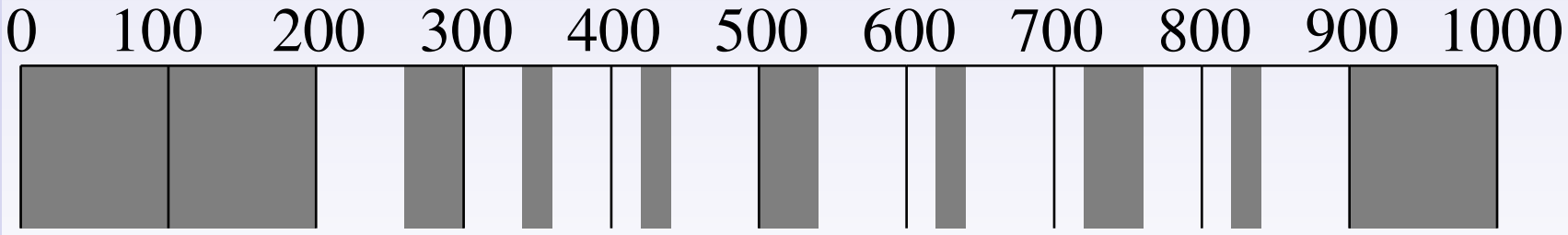
$\Rightarrow$ difficulty in allocating large contiguous chunks of memory

$\Rightarrow$ even if there is enough free memory, an allocation can fail.

To succeed, the free memory must be contiguous (fragmentation).

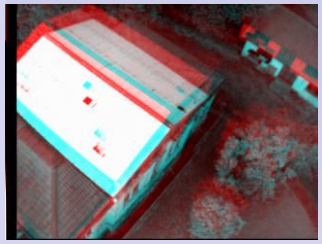
Example:

block size: 20 KB - free memory: 500 KB - request: 100 KB



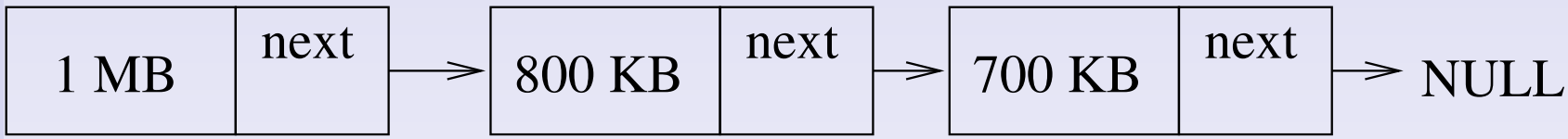
$\Rightarrow$ failure

`malloc-simple` as an alternative to the no mmu malloc version



Large memory allocation under uClinux

Huge memory allocation can be made by building a chain of smaller amount of memory:




```
struct buffer
{char *data;
 unsigned int size;
 struct buffer *next;
};

struct buffer * alloc_mem (unsigned int size, unsigned int *get_size);
```

D. McCullough, Why is Malloc Different Under uClinux?, http://www.linuxdevices.com/articles/AT7777470166.html

5272/5282 differences

- Major difference between 5272 and 5282: supervisor mode
- All peripherals of the former can be accessed by root (user space), the latter requires modules
- Modules use a structure exactly similar to the one found in linux
- SSV provides *with its development board* a module, `ssvhwa.o`, and source code for accessing registers from user space (ioctl calls).

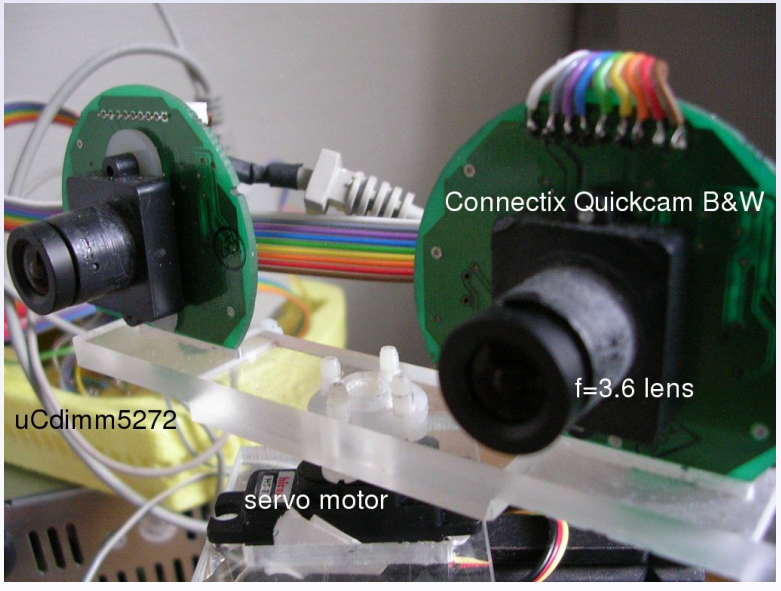
When adding new applications to the image, remember to update your uClinux-dist/uClibc/.config (deduced from config.uClibc in the `vendors` directory):

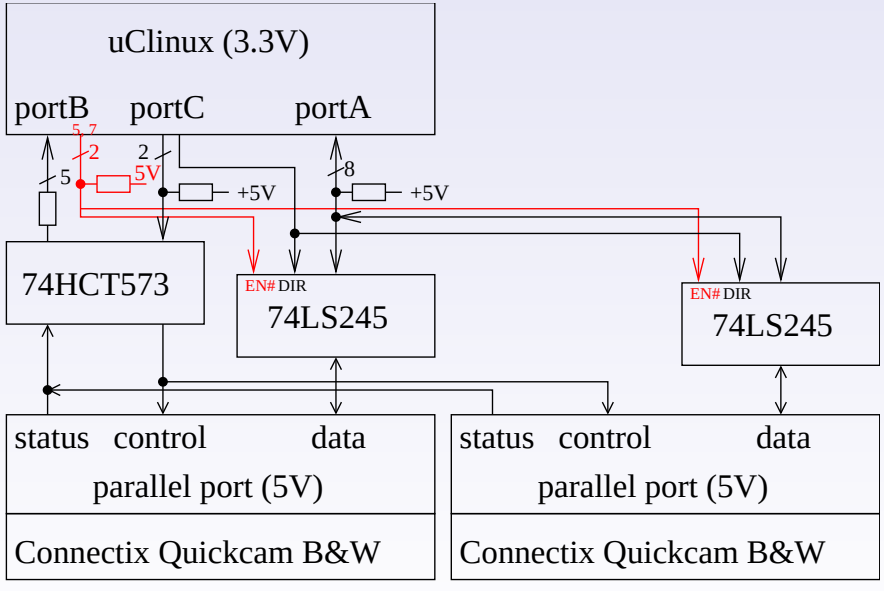
- `UCLIBC_HAS_RPC=y` for NFS mount (client)
- `UCLIBC_HAS_REGEX=y` for `libz` support

B&W webcam

Our first example: remote control of a couple of webcams for stereoscopic viewing

- Old camera (Connectix Quickcam B&W) for connection to a PC parallel (printer) port: the PC is the master, the camera provides the pixel when requested (stored by a PIC on the camera connector).
- Black and white, 6 bits/pixel, max 5 fps.
- Well known protocol available on the web

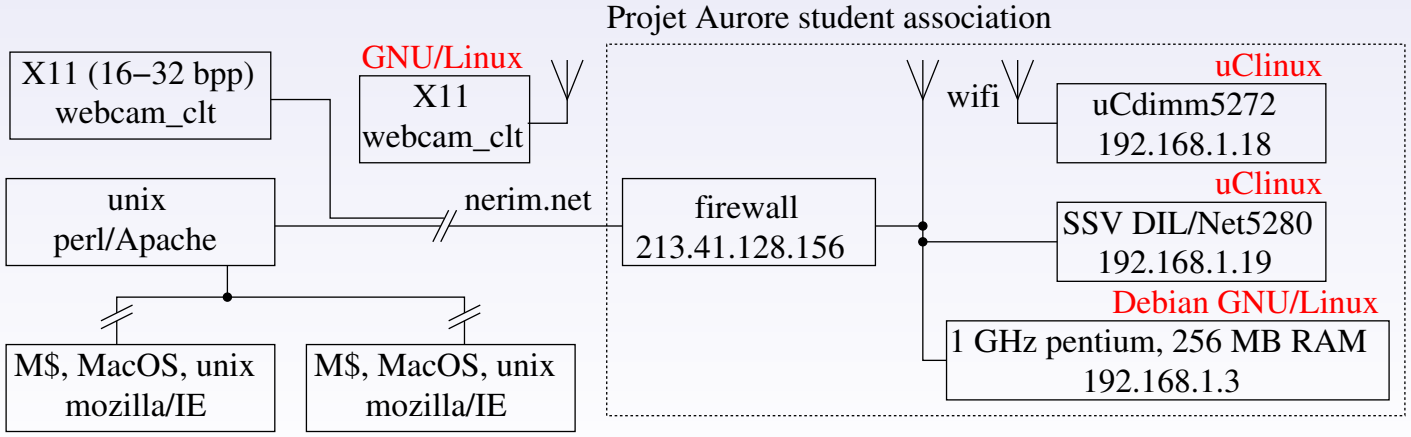




http://www.seattlerobotics.org/encoder/200009/qcam.html

Webcam image transmission

- Multi-threaded (`pthreads`) server on the embedded board, data transfer by wifi/ethernet
  $\Rightarrow$ multiple clients can connect and receive the data.
- Images are transferred compressed (internet) or as PGM (local: 78747 bytes/image, faster on local network than compressing)
- One of these clients is a perl script generating a web page to be broadcasted by Apache (web access to the data: platform independent).
- Remote control of brightness, contrast, pan angle, compression rate (JPEG) +recording

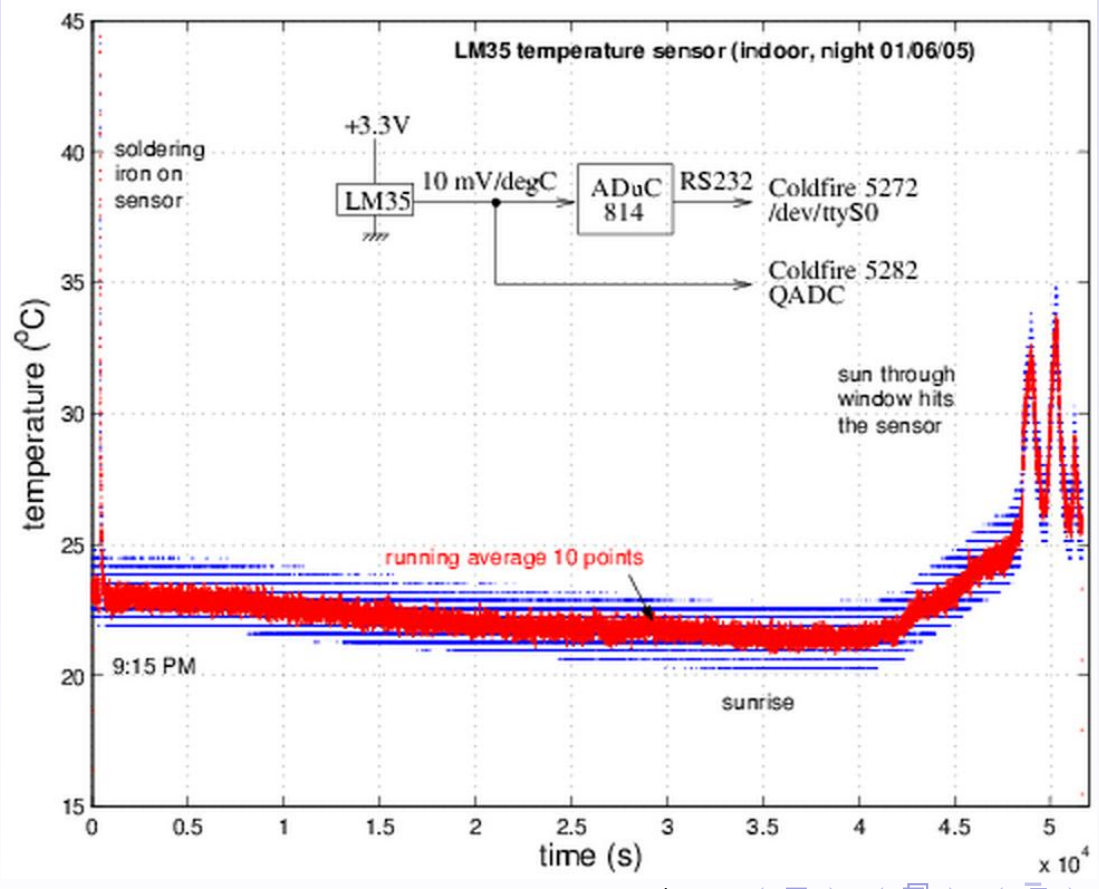


Airborne b&w webcam

Results: small size, low weight and power consumption:
attached to a 2.6 m-diameter tethered ballon filled with He, 1 h autonomy (+digital camera on one RS232 – could have been GPS or ...)

Analog samples

The world is usually analog (temperature, pressure, height, voltage ...)
No ADC on the 5272: an external microcontroller (ADuC81x) connected to a serial port provides the environment monitoring data (temperature) transferred to the clients.



Example: LM35 temperature sensor, 10 mV/degC, sample rate=1 Hz

QADC on the 5282

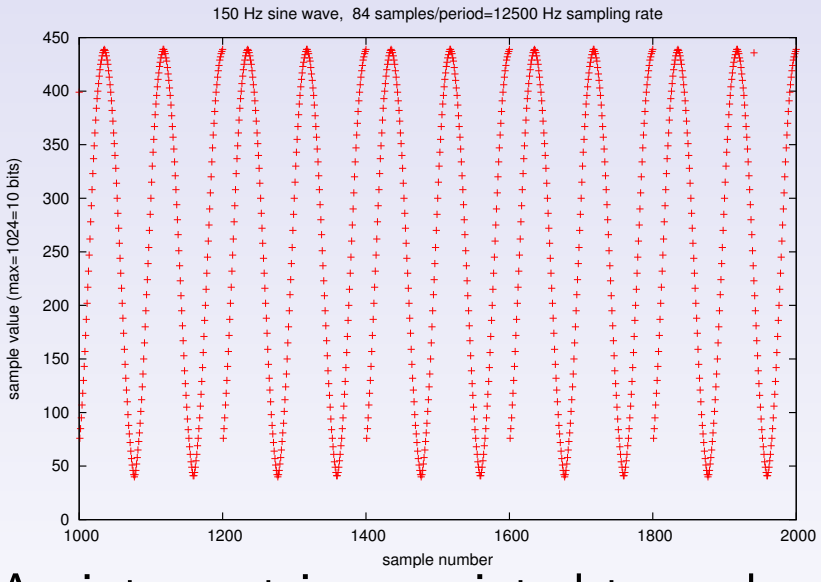
The 5282 Coldfire includes a multiplexed analog to digital converter additionnaly, hardware for queued conversion and possibly additional multiplexing
Basic conversion ($\simeq$ 1 sample/s):

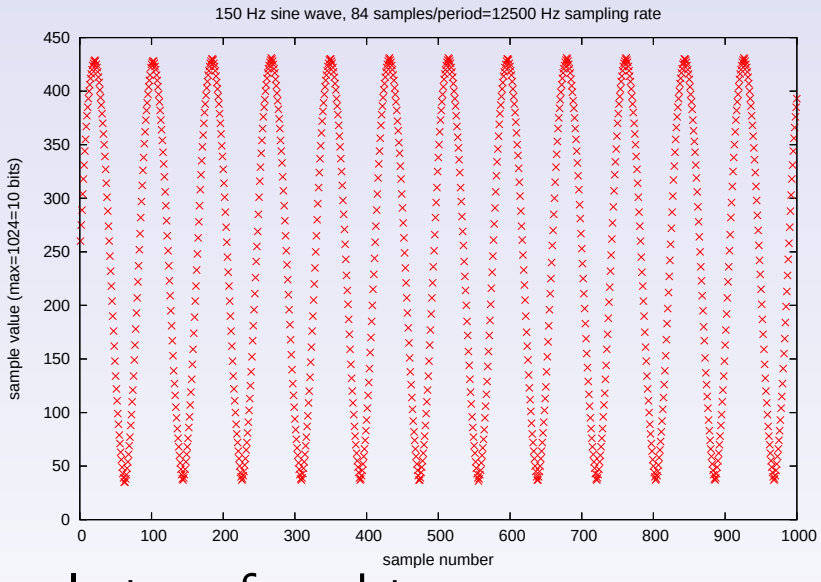
```
*((volatile u16 *)(QADCMCR)) = 0x00; // enable ADCs
*((volatile u16 *)(QACR0))   = 0x7f; // default val = 19
// 52=AN52=PQA0, 53=AN53=PQA1, 62=(VH-VL)/2
for (i=0;i<nbscan*2;i+=2) ssvhwa_write16(CCW+i,0x00C0|52); //queue1 read, 16 ck

*((volatile u16 *)(QASR0))=0x0000;
*((volatile u16 *)(QACR1))=0x2100;   // queue 1 single scan mode, software trig 2100
do {data_w=*((volatile u16 *)QASR0);} while ((data_w&0x8000)==0); // wait for conversion
for (i=0;i<nbscan*2;i+=2) {data_w=*((volatile u16 *)(RJURR+i));   // read result
                  printf("%x=%d/10=degC",data_w,(int)((float)data_w*3.3/1.024));
```

Fast, continuous recording

We want to use the 5282 as a sound card (continuous fast recording of a single channel)
⇒ 12500 Hz max. sample rate, 10 bits/sample, until the memory is filled





An interrupt is associated to each sample transferred to user space: leaves time to multitasking.

Color camera: the OV6620 CMOS sensor

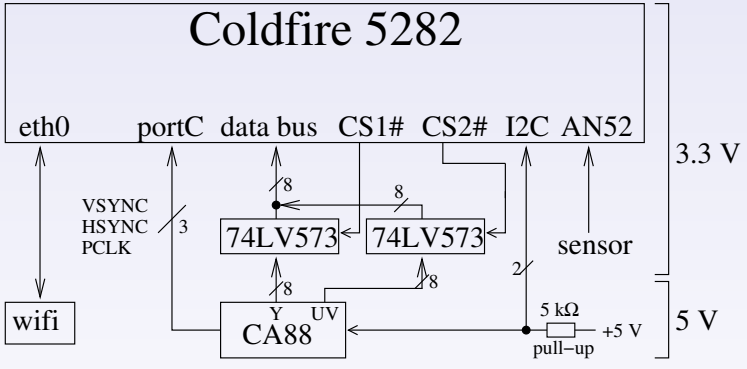
Second application: connect a color camera to the 5282 board
Different challenge: here the camera is the master
50 fps possible, limited by a purely software data acquisition
Omnivision OV6620: available in low volume, low cost (Lextronic CA88), datasheet available on the web

http://www.lextronic.fr/Capteurs/CAMERA.htm

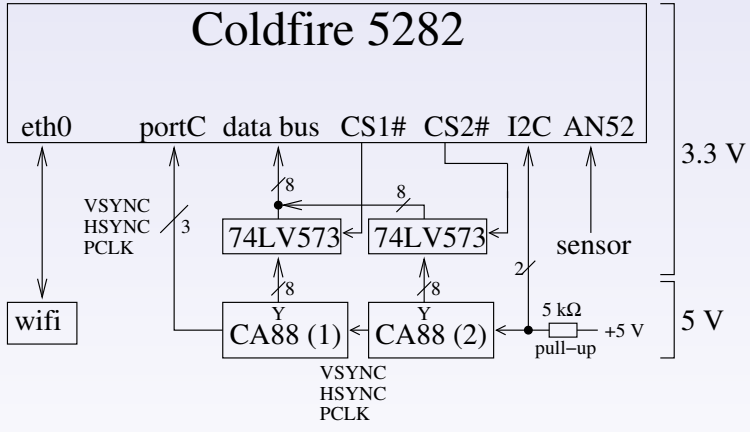
Color camera setup

Program the camera by I^2C protocol (address=0xC0)
Connect the camera to the databus, poll the control signals:
the hardware is the same than the one found on ISA cards,
the adress selector (CS#) is provided onboard by the Coldfire
First operation: decrease frame rate (register 0x11 of OV6620) to be compatible with **software** acquisition

One camera, 16 bits mode

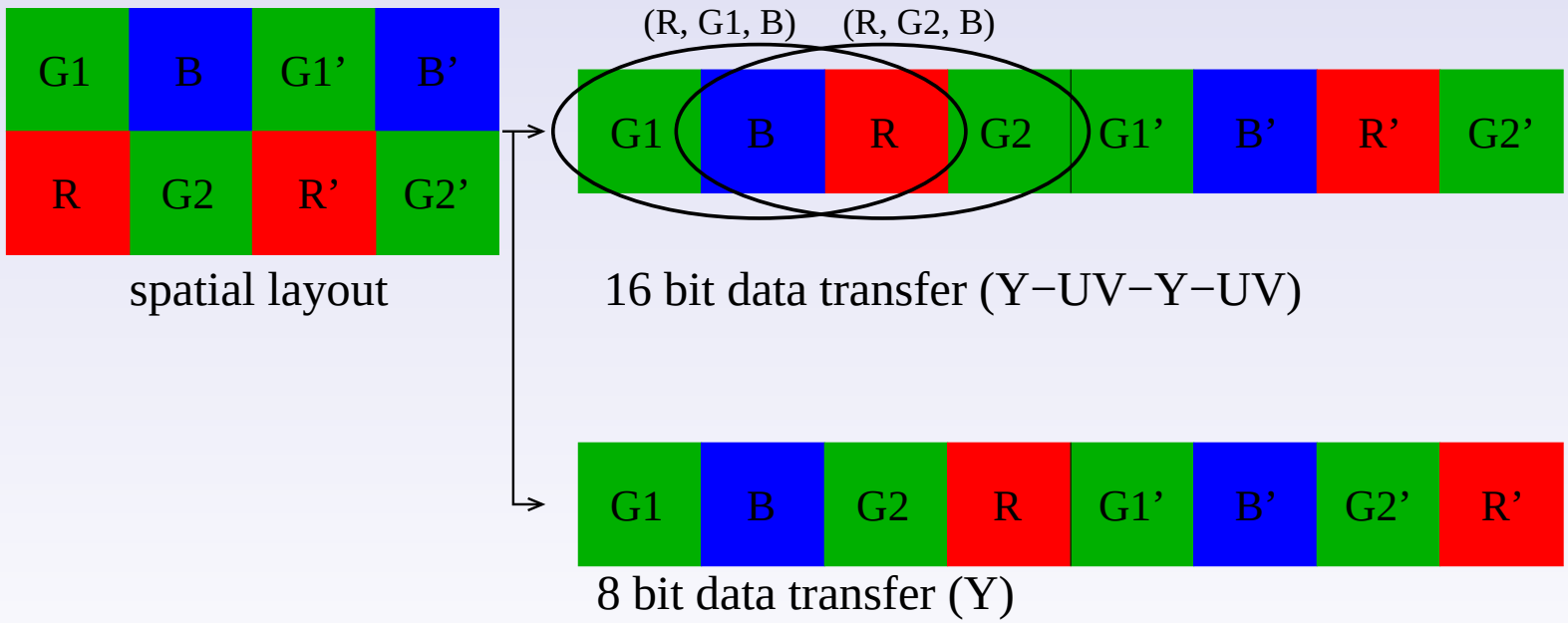


Two cameras, 8 bits mode



Color images

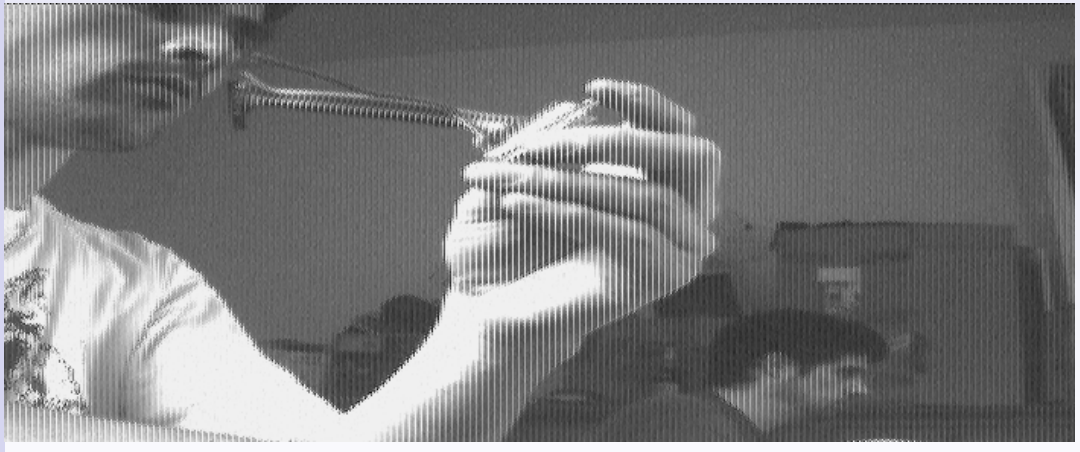
In YUV mode, Y is the light intensity ⇒ black and white image
For color image: 8/16-bit mode, use RGB mode (register 0x12 of OV6620 to set RGB mode, register 13 to set 8/16 bit mode)
How to convert an array of values to color images: raw layout → RGB



Raw data to RGB conversion

8 bit mode:

16 bit mode:

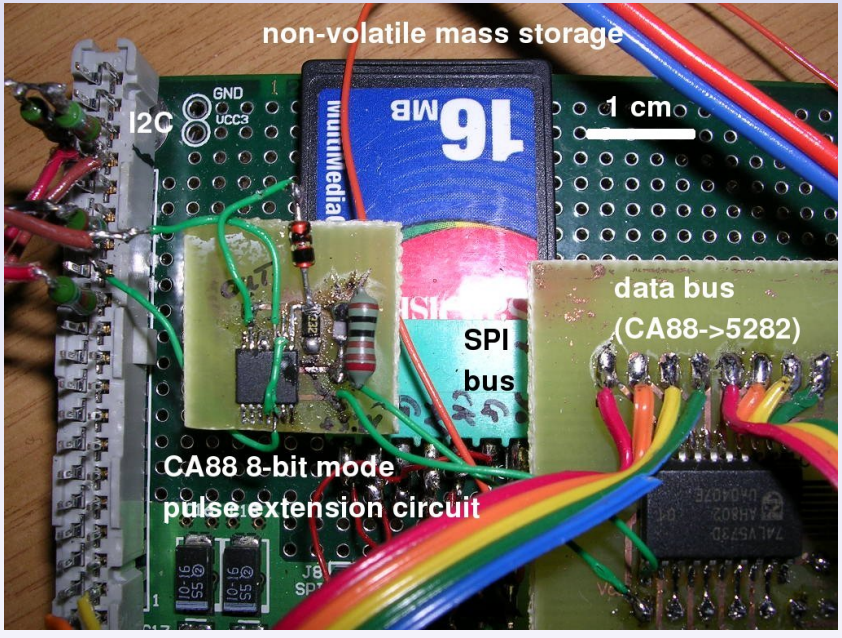


Client/server architecture

- Kernel module on the Colfire 5282 grabs the image (interrupt disabled: `ssvhwa` user space harware access module is far too inefficient)
  $\Rightarrow$ concern of packet losses due to lack of interrupt during image acquisition
- A multi-threaded user-space server `reads` the image and sends it (TCP/IP)
- Non-lossy compression algorithm on the server (`libz`) since the image is not yet organized as a PPM structure (cannot used JPEG here).
  Very slow and depth 1: 202752 $\rightarrow \simeq$175000=15% gain
- A client on the PC side receives the image, converts to PPM and displays (raw X11)

Non-volatile mass storage

Non-volatile mass storage: the MultiMediaCard accessed in SPI mode (3 wire interface)



$\Rightarrow$ mass storage without disabling the interrupts (as is the case with `write/printf` or `NFS`)

Conclusion and perspectives

- Imaging is probably the most demanding application: requires a large amount of RAM, fast CPU, high bandwidth.
- Was successfully implemented on Coldfire based boards running uClinux.
- Disable interrupts for hard real time (grab all pixels)
- Ability to store analog values, continuously at high sampling rate
- Store on non volatile storage support (MMC)

Future activities

TODO:

- release all modules and add documentation
- large audio sample recording+real time decoding of digital data (satellite image decoding) for storage on MMC
- high resolution camera (3 Mpixels)
- install this while setup on a mobile platform (old remote controlled car): issues of power consumption, behaviour hierarchy, wake up upon external event ...

Physics sensing applications:

- proton precession magnetometer (monitoring the Earth magnetic field requires minimal electromagnetic impact but enough signal processing power)
- bat ultrasound recording (requires memory, handheld device: low power consumption/weight)