Report on NGTCCD Workshop mission

San Diego - 9-12 Aug 2005

From: P. Duhoux
To: M. Cullum
Cc: P.Sinclaire, M.Riquelme, R.Reiss, K.Wirenstrand

1 – Purpose of the mission

The mission was aimed to learn about internals of the PMC and TIM boards' embedded DSP software, understand the timings and exercise the interrupts.
This workshop was taking place at ARC's premises in San Diego, USA. Dr. R. Leach was my principal contact.

2 – Test system description

2.1 - Hardware

The NGTCCD systems comprise 3 main parts:

- PMC board ARC-65 installed on the PMC Slot of a MVME2700 CPU board.
- ARC Controller
- Linux Laptop APR2004 – VxWorks 5.4

In addition

- a power supply 24V 3A for the controller
- a Fiber Optics (FO) pair connection between the PMC to the controller
- a Point-to-Point Ethernet connection between the Laptop and the CPU.

The PCI pin **`INTA*`** and the FO **`XMIT*`** and **`RECV`** Strobe signals were used for timing measurements on a logic analyzer.

2.2 - Software

The Linux Laptop has been installed with the VLT Common Software Release APR2004 + patches. The necessary environments were created that allow the CPU to boot from the Linux machine.
The VxWorks device driver **`arcdrv`** latest version 1.14 was used and modified.
This module hosts also the PMC and Controller embedded DSP code.

3 – Activity report and tests

3.1 – PMC code architecture

The PMC code is implemented as an endless loop on FO receive. Host communication via PCI generates an interrupt that allows to process host commands. Commands are processed locally or forwarded to the TIM board depending on the destination flag. During the readout, any TIM command must be avoided since the FO link is then busy with data transfer.
Even though this code remains very complex as it uses specific features provided by the DSP 56301 processor for DMA and PCI interface, its structure is now better understood and code extensions can be considered.

3.2 – Interrupts

On ESO's request 2 distinct interrupt sources have been implemented (controlled by the primitives `IMAGE_IRQ` and `REPLY_IRQ`). In addition to the DMA transfer completion interrupt, a reply ready interrupt is now supported.
Since only one interrupt line (PCI signal `INTA*`) can be asserted by the PMC, a dedicated flag (`INTA_SRC`) has been introduced that contains the code of the interrupter (1 for command, 2 for DMA). A new vector command `READ_INTA_SOURCE` (code `0x8B`) has been implemented that returns the content of the storage location `INTA_SRC`.

On the driver side, the interrupt `INTA*` **must** be disabled and cleared prior to configuration and enabling. Both interrupt vector and level take the value `0x19` = `25.`
The Interrupt Service Routine (ISR) `arcdrvISR()` must retrieve the interrupt source and clear the interrupt before giving the semaphore `arcdrvIntSem` to the interrupt task `tintARC` for DMA interrupts; for CMD interrupts, the ISR gives the semaphore `arcdrvCmdSem` directly to the driver function `arcdrvCommand()` pending on it.
The interrupt task `tintARC` is an endless loop waiting on the semaphore `arcdrvIntSem` and invoking a user-function whenever available.

Only the DMA completion interrupts have been tested so far.

Tests have shown that most of the commands do not need delay waiting for the reply: solely the commands **PON**, **POF**, **CLR**, **SBV** and **SGN** need a significant delay as the associated operation involves HW settings on the Controller. But these commands are part of the system initialization and thus do not jeopardize the run-time performances.
For the commands expecting data from the controller a short delay of 100 to 200 μs is necessary until data is available in the DSP output FIFO.

command	Delay [ms]
PON	<500
POF	0.6
CLR 1	1.8
CLR 3	6.0
SBV	4.2
SGN	0.6

Linear approximation of the execution time of the command CLR (Wipe):

`CLR n` : $2.67 \times n - 1.25$ ms (for the chip CCD57-10).

3.3 – PMC code possible enhancements

The DSP 56301 is a very fancy and powerful processor running at 100MHz (10ns / instruction). In addition to its on-chip memory 3 banks of external memory (8MB in total) can be addressed.

3.3.1 – Image data unscrambling

The image data is copied into the CPU DRAM via DMA. The PMC is DMA controller. Hence the CPU is not affected by the transfer, only the PCI bus is active. As a consequence of the readout sequence, the pixel data is copied as readout in to the CPU DRAM: that means that the image data coming from output Right is mirrored and interlaced with data coming from output Left. So far the CPU takes care of re-ordering the pixel data as a function of the setup. This pixel manipulation however is not optimal for the CPU as it must handle 16bits integers. A more elegant and performing solution would consist in migrating this operation onto the DSP. As illustrated in section 3.5 the data transfer via the FO from the Controller to the PMC takes place at a speed of 1 pixel / µs into the receiving FIFO. In addition a time gap of at least 5µs is preserved between 2 lines. Hence it shall be possible to redesign the data transfer so as to hook an interrupt on FIFO Half-Full, copy the 512 bytes of the FIFO into a double DRAM buffer, re-organize the pixels in the data buffer and send the data via DMA to the host DRAM without affecting significantly the data transfer time.

FIFO Half-Full Interrupt Service Routine
```
append FIFO data to DRAM Buffer #b
if (DRAM Buffer #b full) then
      set data available #b
      swap buffer
endif
```

Data Transfer activity
```
allocate 2 DRAM buffers of N lines each
do
      while (!data available #b) wait
      re-organize pixels in DRAM buffer #b
      initiate DMA transfer
while (!all pixels sent)
```

This re-design would also permit to physically split the window buffers and transfer them into 2 different image buffers on the CPU.

3.3.2 – Image processing

Using a similar construction, it shall be possible to perform part or whole of the image processing for guiding purpose (may be restricted to windowed readout only and limited to a maximum window size).

3.4 – Multiple window readout

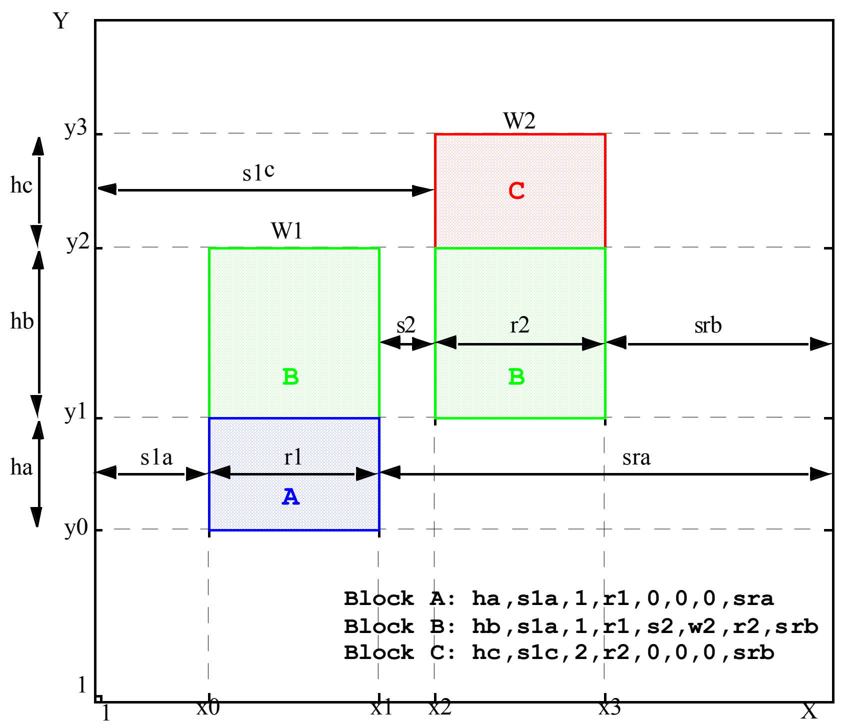
The implementation of this feature has been discussed on Friday morning and requires a partial redesign of the TIM board embedded DSP code in order to support vertically connected windows (not overlapping). A rough analysis shows that any window combination can be mapped as a sequence of 1+3w+1 integers:

$$\mathbf{n}\ [\mathbf{s}_i\ \ \mathbf{w}_i\ \ \ \mathbf{r}_i]^{1..w}\ \mathbf{s}_r$$

where

w	number of windows
n	number of rows for this entry line applicable
s_i	number of pixels to skip before window #*i*
w_i	window index for data
r_i	number of pixels to read for window #*i*
s_r	number of pixels to skip at end of line

The following Figure 1 illustrates such a block structure:



```
Block A: ha,s1a,1,r1,0,0,0,sra
Block B: hb,s1a,1,r1,s2,w2,r2,srb
Block C: hc,s1c,2,r2,0,0,0,srb
```

Figure 1 - Windowed readout

The corresponding readout sequence would look like:

```
FRAME TRANSFER
SHIFT Y0 LINES DOWN
DO    B,L_BLOCK
READ NEXT BLOCK ENTRY
DO    N,L_LINE
DO    W,L_WINDOW
DO    si
SKIP
DO    ri
READ                      ; to Window #wi
```

```
L_WINDOW
      DO    sr
      SKIP                    ; until end of line
L_LINE
      SHIFT 1 LINE DOWN
L_BLOCK
```

3.5 – Timing

Using the logic analyzer to monitor and record the lines **RECV-Strobe**, **XMIT-Strobe*** and **INTA*** it was possible to assess accurate timing measurements for a window 20×20 located at [200;200]; Exposure Time=0ms, no wipe between images, output __L only (Chip setup CCD57-10 with 528 lines of 560 pixels):

Figure 2 illustrates the overall timing of 1 exposure. The time T_0 corresponds to the frame transfer + skip 200 lines and is well in accordance with previous measures (see report May 2005).
Immediately following the reply to the command **SEX**, the command **RDA** is sent by the controller and instructs the PMC to prepare for receiving data.
The time T_1 is the effective readout time.
The time T_2 is the time spent between the reception of the reply to the command **RET** (Read Elapsed Time) and the invocation of the command **SEX** (Start Exposure). It is by far much larger than the theoretical time between commands as required by the PMC embedded software.

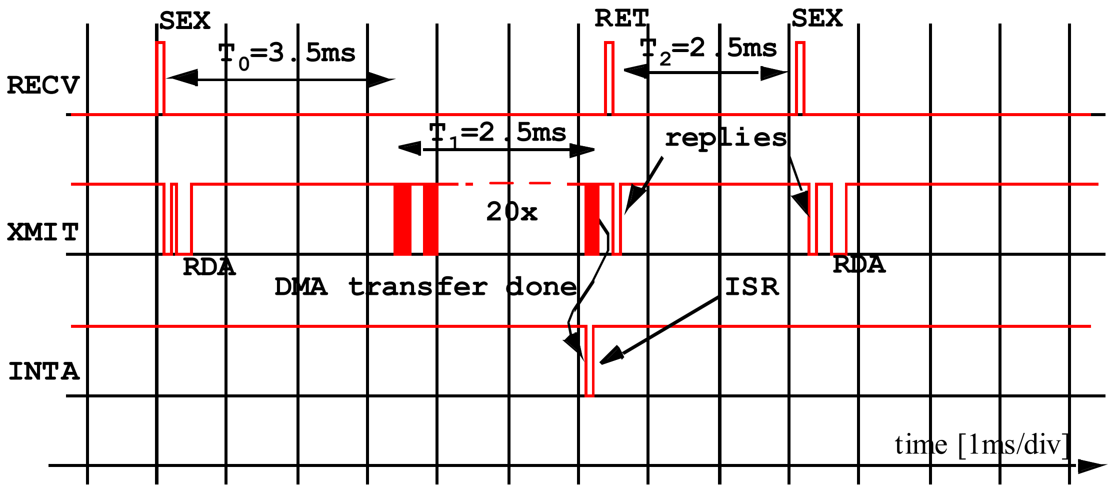


Figure 2 - Timing 1ms/div

Figure 3 shows a close-up of the above timing. It displays the transfer bursts of each 20 pixels as of the readout. Each burst is 20µs long corresponding to the readout time. The interrupt **INTA*** is asserted immediately after sending the last pixel. The time between bursts corresponds to skipping the remaining pixels of the line (560-220=340), shifting 1 line down, and skipping the first 200 pixels of the line (approx 120µs).

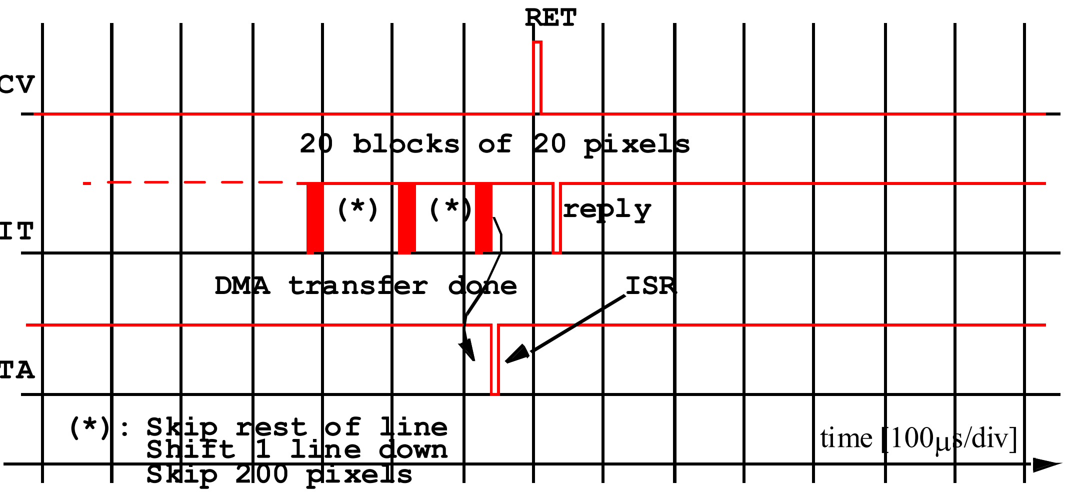


Figure 3 - Timing 100µs/div

Figure 4 illustrates the detailed timing between 2 images. The **`INTA*`** signal is asserted for 10µs (cleared by the **`ISR`**). The interrupt task **`tintARC`** being waken up on semaphore by the **`ISR`** returns in less than 20µs, the next command **`RET`** can then be issued approx. 35µs after the interrupt has been asserted.
Following, the next **`SEX`** command can now be issued less than 125µs later as a result of the fine tuning of the reply waiting loop. Active polling based on the construction:

```
while (reply not ready)
      while (delay>0)
            for (i=N;i>0;i--);
            delay--;
      endwhile
endwhile
```

where *N* defines the duration of the active wait. Formerly set to 1250, it could be reduced to 46; so that the delay can be expressed in micro-seconds.

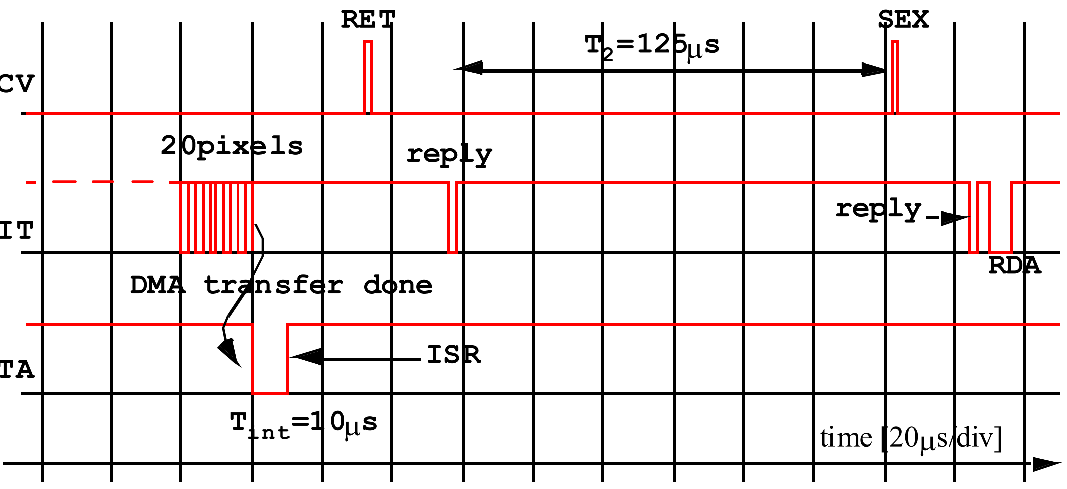


Figure 4 - Improved timing 20µs/div

The following performances could be achieved:

Timing Performances

Window [20x20]	CCD57-10	Unit
WIPE CHIP	<5	ms
FRAME TRANSFER	<3	ms
SHIFT 1 LINE DOWN	<5	µs
SKIP 200 LINES	<1	ms
SKIP 200 PIXELS	≈50	µs
READ 20 PIXELS	≈20	µs
TIME to NEXT IMAGE	≈120	µs
READOUT Left	≈6.5	ms
Max Frequency	≈150	Hz
Full Frame Left	≈3.3	Hz

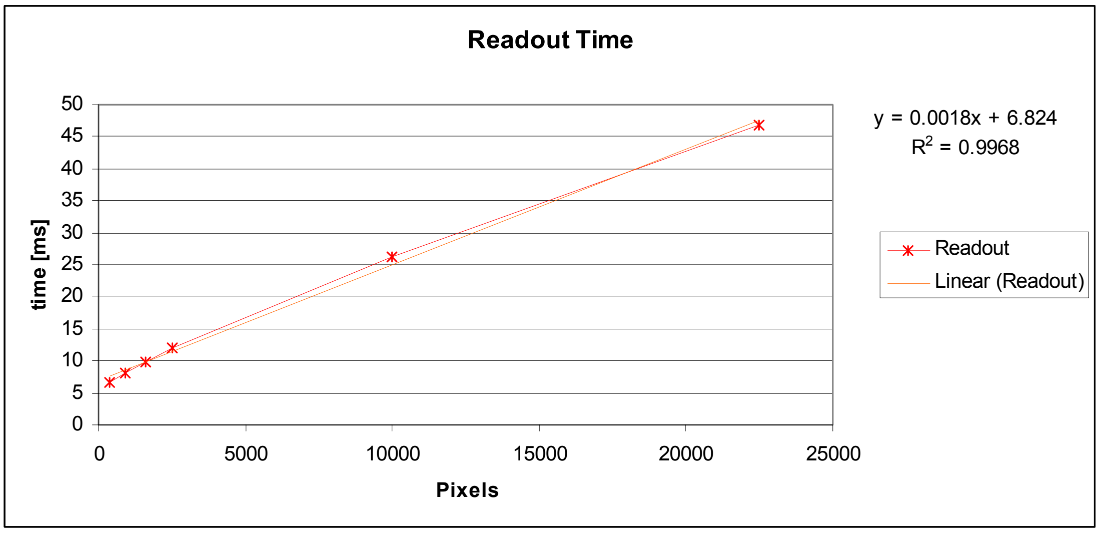
1000 Frames
No Wipe between exposures
ExpTime=0ms
Location: [200;200]

From RET to SEX

Table 1 – Performances

The following diagram shows the behavior of the pure readout time as a function of the number of pixels readout. All windows are square and located at [200;200]. The exposure time is 0ms with no wipe between exposures using output Left only. Measurements have been done on loops of 1000 images.

Window	Pixels	Readout Time [ms]	Cycle Time [ms]
20 × 20	400	6.5	6.9
30 × 30	900	8.1	8.5
40 × 40	1600	9.8	10.2
50 × 50	2500	11.9	12.4
100 ×100	10000	26.3	26.7
150 ×150	22500	46.8	47.3
Full Frame	296736	302.2	302.8



4 – Conclusions & Future activities

Even though the beginning of the mission did not look too promising, as almost 2 days were spent in getting the system acquiring images, many questions, issues could be addressed which helped a lot getting a better understanding of the overall architecture, function, data flow.
Time could also be found to discuss possible enhancements and their implementation.
The major concerns are related to the actual development environment (DSP assembly code) and the complexity of the DSP 56301 (embedded PCI interface, embedded DMA controller).

The next activities will aim at:

- Implementing the 2 windows mode readout.
- Establish the formula to estimate the effective exposure time and readout time as a function of the setup.
- Investigate feasibility and cost of data unscrambling on DSP
- Investigate feasibility and cost of data processing on DSP

___oOo___