Archimedes <-> Raspberry-Pi Bridge

Alain Lowet - 2023

Goal:

The project involve the building of a link between an Acorn ARM Risc computer (ARM300,300,500,4000,5000 RISPC…), based on DIN41612 backplane to interface with a Raspberry-Pi SBC through the use of a dual port ram, permitting asynchronous reads and writes between the two platforms. The RPI card will be included inside the Acorn computer.

Card capabilities:

The size of the double port ram, who act as a memory buffer between the ARM and the RPI, is 4K * 8 bits, covering the maximum space a byte based podule card can cover.

On the software side, the idea is to use this card as a real bidirectional bridge between old Acorn ARM based machine and new RPI ARM technologies. Meaning, on the Acorn ARM machine point of view, the capability of using the board to access peripheral located on RPI side, examples are:

1. Modified RAMFS to expand memory with RPI memory
2. An FS to read and write on RPI USB devices
3. Use RPI as co-processing unit
4. Use ethernet capabilities of RPI
5. Any way to transmit video information to replicate on RPI HDMI port…

On Raspberry-Pi side, card can be used for:

1. Capturing keyboard and mouse coordinates from Acorn ARM machine
2. Usage of Acorn ARM disquette for reading and writing old software directly from RPI
3. Usage of HDD peripherals for reading and writing old software directly from RPI
4. Link to other specific podule present in the machine (e.g. midi I/O podule)
5. Link to Acorn standard Econet module

Technical choices:

A minimalistic approach was chosen to simplify a maximum the electronic interface.
The double port ram is based on Renesas IDT 7134 4K* 8bits Static SRAM.

On podule side (DIN 41612), the SRAM is directly linked to podule bus, without any address arbitration for the initializing sequence (meaning this part will be handled by a specific boot process explained later in this document)

On RPI side, GPIO ports of the raspberry pi are linked to SRAM by the use of three TXS0108E 8-Bit bidirectional level shifting voltage translators. A serial I2C eprom is also present to permit auto-detection of the podule as a HAT interface on RPI side.

Power supply is handled by a simple 5v - 3.3v voltage converter LM1117 capable of delivering up to 800mA at 3.3v, this much more than sufficient for powering the TXS0108E level translators. 5v power supply is taken directly from the podule backplane. **CAUTION!**, this is also the way the RPI will get it’s power, so it is important to verify that the computer power supply can deliver at least 2.2 Amps/5V only for the RPI.

Software description:

Initialization procedure

During the boot process of an Acorn ARM computer, the device checks for the presence of podules plugged into the backplane. To be detected, a podule needs at least to provide basic information at startup by setting up a specific byte at address 0 of the podule space at booting phase.

Because of the minimal electronic interface option chosen, the IDT 7134 double port ram needs to contain this information at startup. For this purpose, during initialization phase of the RPI SBC (remember that the board is switched on at the same time as the Acorn computer), the /reset line of the ARM computer is maintained low until RPI has written the starting byte into address 0 of the SRAM. When done, /reset line is released and Acorn machine is allowed to boot. During the process, the machine will retrieve the byte written at address 0 and configure the podule properly.

This is the very minimal basic procedure to implement, but due to this flexibility, any other way of booting the podule can be used (usage of extended podule ID mode -> see Acorn podule documentation).

Pseudo code:

RPI initialization side:

1) computer switch on
2) set R/OE (GPIO27) HIGH
3) RPI GPIO02 port (pin 3) set as output, LOW level (/reset Acorn ARM)
4) RPI write id information byte to address 0 of IDT7134
    a) setAddress
    b) setData
    c) set R_R/W (GPIO04) LOW
    d) set /R_CE (GPIO03) LOW
    e) wait
    f) set /R_CE (GPIO03) HIGH
    g) set R_R/W (GPIO04) HIGH
4) RPI GPIO02 port (pin 3) set as output, HIGH level (release /reset Acorn ARM)
5) RPI GPIO02 port (pin 3) set as input

ARM initialization side:

1) computer switch on
2) Podule id is read during initialization and configured as expected…

Reboot procedure Acorn ARM computer side:

After first initialization process (Machine switch-on), the /restart line (GPIO02) was set as input, it is up to the RPI side to check this pin (polling) for a LOW state.
If a LOW state is detected (meaning someone hit the reset button on the computer), the RPI needs to re-enter the initialization process by relaunching the initialization procedure, forcing the / reset line low until initialization procedure is done.

Reboot procedure RPI SBC side:

Rebooting the RPI side is a little bit more complex, because we need to avoid that at RPI reset, if the computer is still active, RPI does not reset the host computer.

Proposed solution:
?

Computer switch-off, safe procedure:

Because the switch-on , switch-off actions are common between the Acorn ARM computer and the RPI SBC, it is important to follow a clean switch-off procedure to avoid any SD-Card corruption on the RPI side. Therefore, a switch-off procedure will be provided on Acorn ARM computer side to send a message to the RPI via the bridge to intimate a clean shutdown of the RPI before a switching-off acknowledgment message is sent to the user. This must be part of the software bridge functionalities (ARM & RPI side)

Bridge protocol:

The working of the bridge will be based on a message based protocol, with id's of messages sent by one side, and acknowledged on the other side. Therefore, the 5 last addresses will be used as mailbox between the two systems, all the rest of the memory spaces will be useable for specific tasks as requested and activated by one side or the other.

SRAM capabilities:

The Renesas IDT 7134 SRAM does not have any internal ways of handling the messaging process between the two ports, it is up to the software process to handle the data and organize a valid arbitration for accessing the SRAM space

So, the last bytes of the SRAM space will be used to store a mailbox system that will manage the arbitration logic for each tof the two sides (ARM/RPI).

Practically,

Address 0x0FFB: Failure byte detail
Address 0x0FFC: OPID byte of operation requested by ARM to RPI
Address 0x0FFD: Acknowledgement status for operation requested by ARM to RPI
Address 0x0FFE: OPID byte of operation requested by RPI to ARM
Address 0x0FFF: Acknowledgement status for operation requested by RPI to ARM

Acknowledgement statuses:

0x00: No operation handled
0x01: Operation currently serviced
0x02: Operation handled with success
0x03: Operation failed (check Address 0x0FFB for failure detail)

Arbitration logic:

ARM -> RPI:

a) 0x0FFD = 0x00 ==> ARM can send new request to RPI (write new OPID to 0x0FFC)
b) 0x0FFE != 0x00 ==> ARM execute request and set 0x0FFF to 0x01
   If operation completed with success ==> set 0x0FFF to 0x02
   If operation completed with failure ==> set 0x0FFB to failure code detail
   ==> set 0x0FFF to 0x03

RPI -> ARM:

a) 0x0FFF = 0x00 ==> RPI can send new request to ARM (write new OPID to 0x0FFE)
b) 0x0FFC != 0x00 ==> RPI execute request and set 0x0FFD to 0x01
   If operation completed with success ==> set 0x0FFD to 0x02
   If operation completed with failure ==> set 0x0FFB to failure code detail
   ==> set 0x0FFD to 0x03

RISCOS- RPI : C Code GPIO configuration