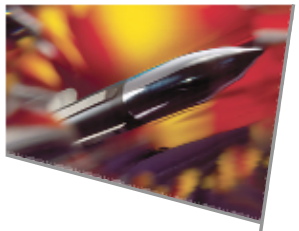
Obsolete processor replacement using an Ada/firmware co-design

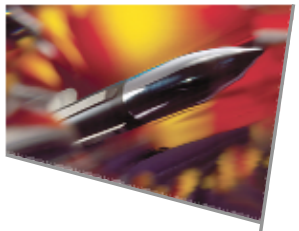
Rod White



Agenda

- Background
  - Problem
  - Asset value
- Options
- Down-selection
- Implementation
  - Approach
  - Details
  - Results
- Conclusions

MBDA MISSILE SYSTEMS

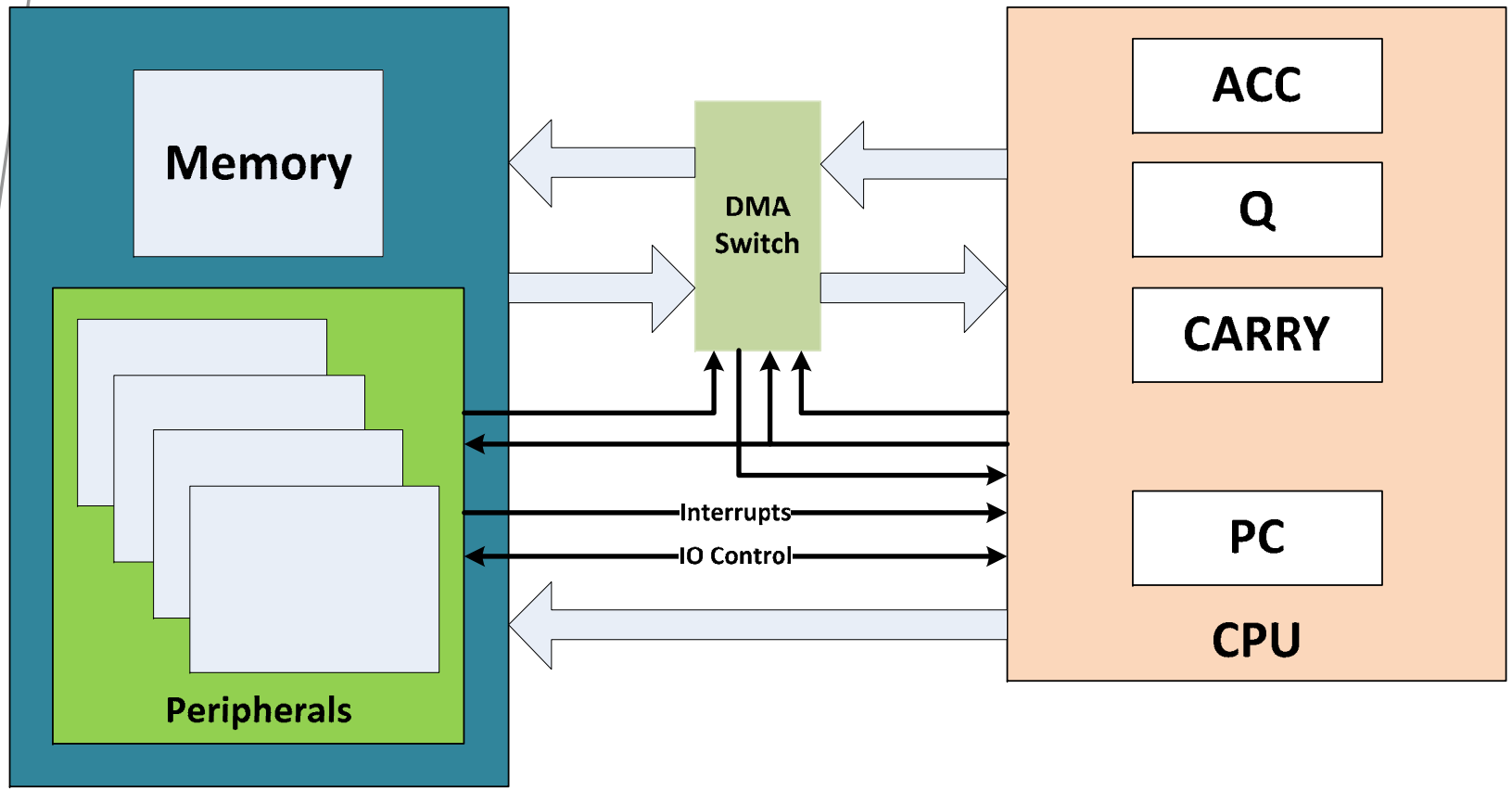


Background – the problem

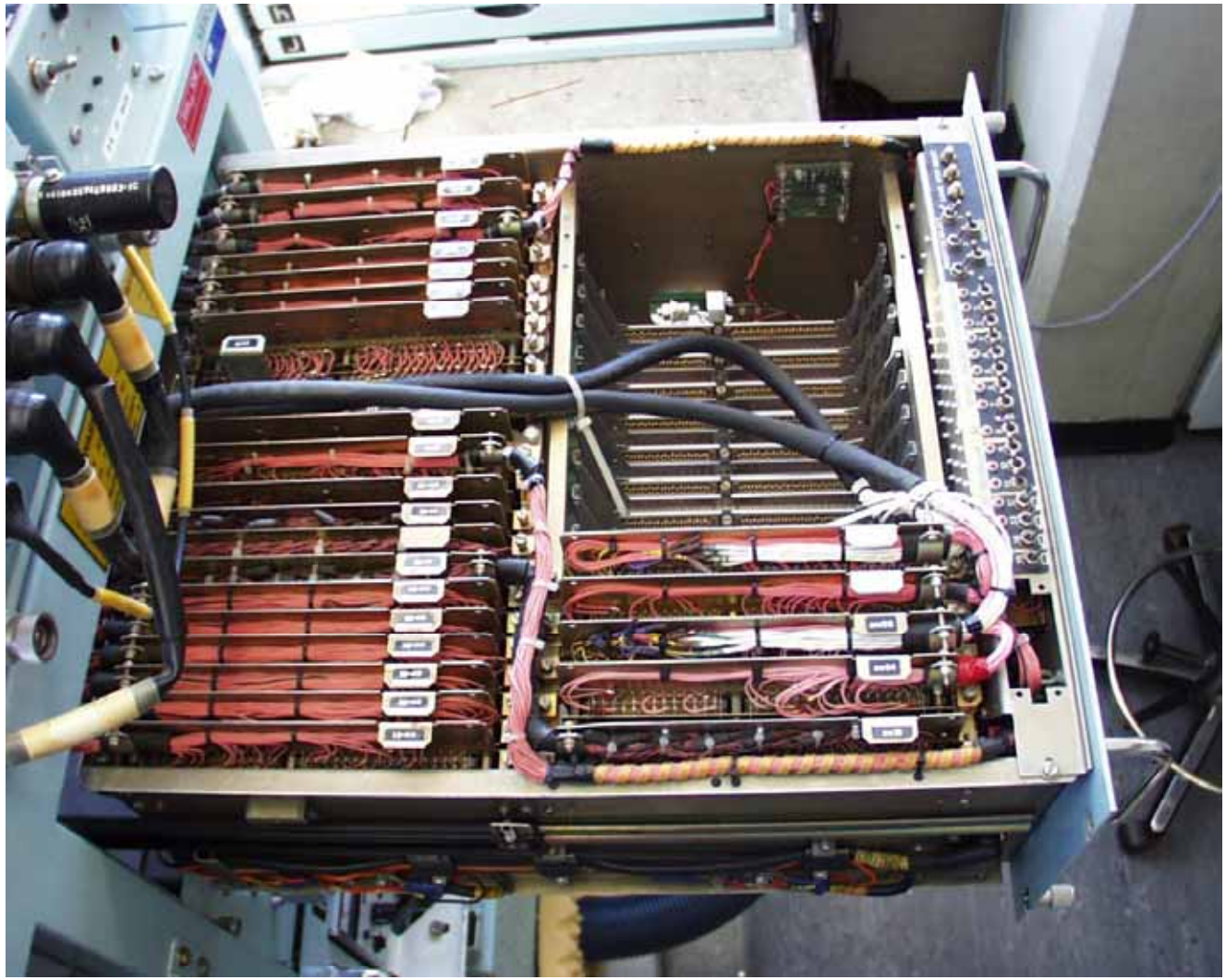
- Over the past few years we, together with a customer, have been investigating (and in some cases implementing) ways of replacing obsolete parts of a test equipment controller and its associated instruments to retain in-service capability into the future. This is a pro-active obsolescence management activity
  - The CPU part of the computer that controls the test equipment is one of the items on their list…
- The test equipment contains a rather old processor that was initially developed in the mid ’70s based largely on TTL logic
  - “Simple” state-machine – micro-code used to implement some higher level (macro) instructions
  - 16 bit architecture
  - 3-7µs macro instruction cycle
  - Multiple busses (input, output, IO instruction/address)
  - DMA – but not as we know it!
  - Two levels of interrupt – effectively clock + others (about 15 sources)
  - … by modern standards not particularly fast or complex machine

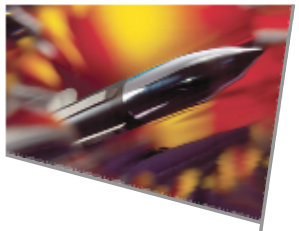
MBDA MISSILE SYSTEMS

Background – computer architecture



Background – the “physical” hardware



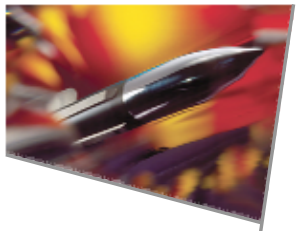


Background – value of the assets

- Need to protect the value of the assets – the "software" elements that run on this equipment

- Two most valuable assets
  - 40+ application test packages (ATP): ~200my of effort
    - Individually relatively simple on the whole but significantly complex in some areas
    - Origins of some of these mid '70s
  - Control software: ~45my of effort
    - Significantly complex, hard real-time (it works because it works…)
    - Assembler implementation – no viable HLL available
    - Developed over many years to a changing requirement
    - Stable – no recent changes
  - The tools to support both of these have been migrated to modern platforms and the assets can continue to be maintained

MBDA MISSILE SYSTEMS

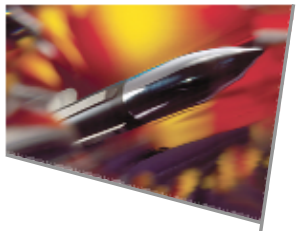


Background – value of the assets

- Test equipment hardware also has significant value, but is possible for parts to be replaced as required, limiting the cost of each step
  - Disc drives and interfaces already replaced
  - DVM and other instruments have been replaced
  - … both of these need to be retained

  - No desire to replace the hardware platform in its entirety – but progressive replacement of components is an acceptable (desirable) approach as it allows elements to be tackled largely according to need and available budget

MBDA MISSILE SYSTEMS



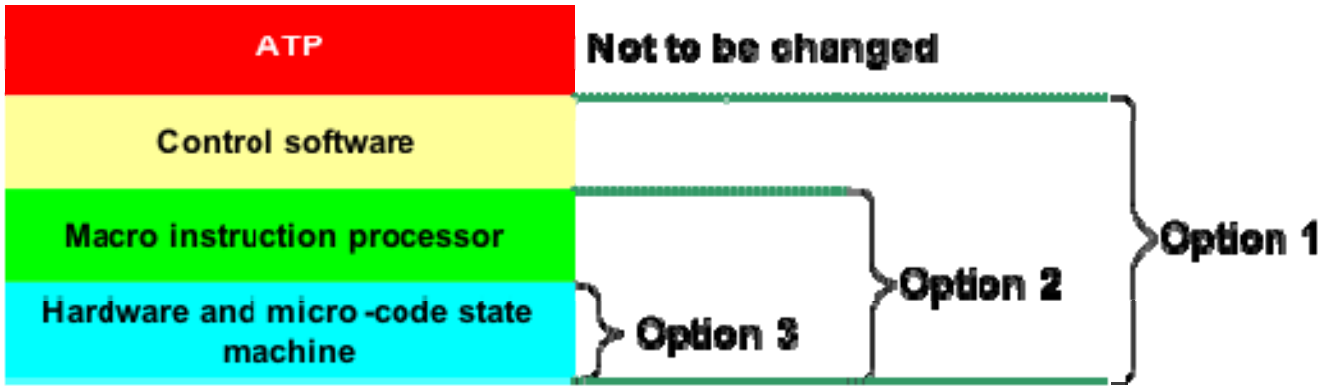
Options (1)

- The fundamental requirement is that the current application test packages must execute unchanged on any replacement platform.

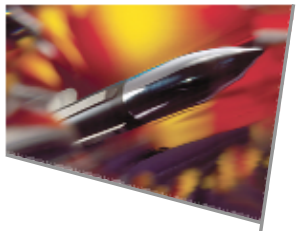
- Three options for the replacement of the controller have been considered:
  1. Replace the computer and the control software that interprets the ATPs and controls the instruments etc.
     - Probably requiring the development of a specialised interface to control the existing peripherals
  2. Replace the computer providing a macro instruction compatible platform for the control software and hence the ATPs
     - Including the replacement of some of the peripherals with capability in the new platform – obsolescence management opportunity
  3. Replace the computer at the lowest level providing a micro-code compatible environment
     - No changes to the existing peripherals – all used exactly as is.

MBDA MISSILE SYSTEMS

Options (2)



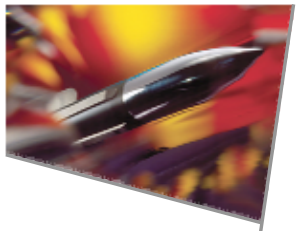
- All peripherals are driven using macro instructions – this layer has to be replaced to allow fundamental changes to the peripheral configuration and a radical approach to their replacement
- The most complex layer by far is that of the control software – simply due to the number of lines of code and its real-time behaviour
- Micro-code is small but has very strict timing constraints – cycle time less than 10ns



Down-selection – rejected approaches (1)

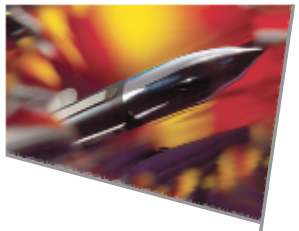
- (1) Replacing the computer and the control software not seen as being a sensible way forward
  - Control software is a large assembler program which is in places extremely complex/"clever" (started in the mid '70s, largely completed in the late '80s)
    - Self modifying code
    - Carefully coded segments to provide specific delays and extreme optimisation
    - Design information/knowledge scarce
  - Unlikely to be able to produce a replacement hardware solution that provides the correct performance
    - Certainly for any reasonable cost
    - Very large effort in testing would be required – though we wouldn't know exactly what results to expect

MBDA MISSILE SYSTEMS



Down-selection – rejected approaches (2)

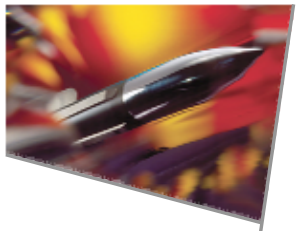
- (3) Replace the micro-code intimately associated with the bus, IO and DMA timing
  - Again detailed design information is scarce
  - Bespoke hardware/firmware design probably required for consistency of behaviour
  - No opportunity to remove some of the “simpler” peripherals
    - They would remain a live obsolescence issue



Down-selection – the chosen approach

- (2) Replacement at the macro instruction level seen as the best compromise
  - Macro instructions are relatively well understood, in terms of both timing and function
  - Will allow the control software to be used unchanged
  - Allows for a software-centric solution
    - In terms of the emulation of the instructions – in effect creating a virtual machine
    - Bus signal timings – no need for complex FPGA logic to emulate these
  - Some peripherals can be replaced: e.g. VDU, Real-time clock, result printer, memory
    - Helps address the obsolescence problem in other areas
  - Should be a relatively simple hardware solution to drive the other peripherals over the busses
    - No more than a simple buffer and voltage translation card is required for 3.3↔5v – TTL is rather old technology compared to modern components
    - Already done as part of a previous task – bus characteristics are relatively well understood from empirical evidence

MBDA MISSILE SYSTEMS

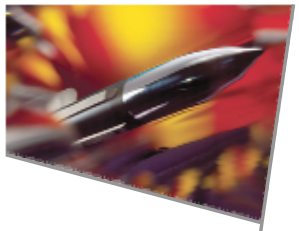


Implementation schedule – "success“ criteria

- Task of many phases – goal to spread over several years planned vs. reactive obsolescence
  1. Produce a prototype virtual machine for the macro instructions (inc. some peripheral replacements) – start to execute ATP and produce the correct display on the standard test equipment monitor
  2. Interface the busses – ability to drive key non-DMA peripherals
  3. Exercise static test package full implementation other than DMA, package – behaves as on original test equipment
  4. Introduce DMA – ability to execute any ATP
  5. Productise interface – make it fit the environment and be a bit more robust
  6. Final acceptance and deployment

- Overall programme spread over ~4 years – steps at ~9 month intervals
  - No exactly that “clean” but largely the approach taken

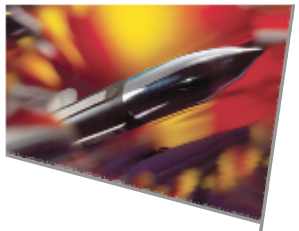
MBDA MISSILE SYSTEMS



Implementation – approach

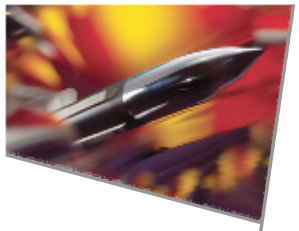
- Solution based on Xilinx Virtex-4 FPGA with single embedded PowerPC 405
  - Xilinx ML403 development card + simple interface card
- Software emulation of:
  - Macro instructions – 60 + device specific IO instructions (67)
    - Function
    - Timing – in some cases variable depending on a variety of conditions
  - Peripheral interfaces/device emulation for
    - VDU (UART interface only – real “monitor” used)
    - Controller front panel
    - Real-time Clock (full peripheral)
    - DMA channel control
    - Result printer interface
  - Interrupt handling and vectoring

MBDA MISSILE SYSTEMS



Implementation – software

- Emulation written in Ada (but could easily be ported to other languages)
  - Used GNAT compiler targeted to bare-board PowerPC
  - Zero-footprint runtime - this is a simple application, no tasking or similar
  - Spark Ada
  - Total size (code + instruction decode tables) : 10.5kBytes
    - Small enough for the code to be fully cached – means consistent execution times as it is forced into cache at startup
- Software managed timings for key bus signals related to IO devices - accurate sequence 500ns pulses across several lines
- DMA signal timing to meet delivery rate of peripheral device (emulated disc drive) – used dead time at the end of instructions cycles

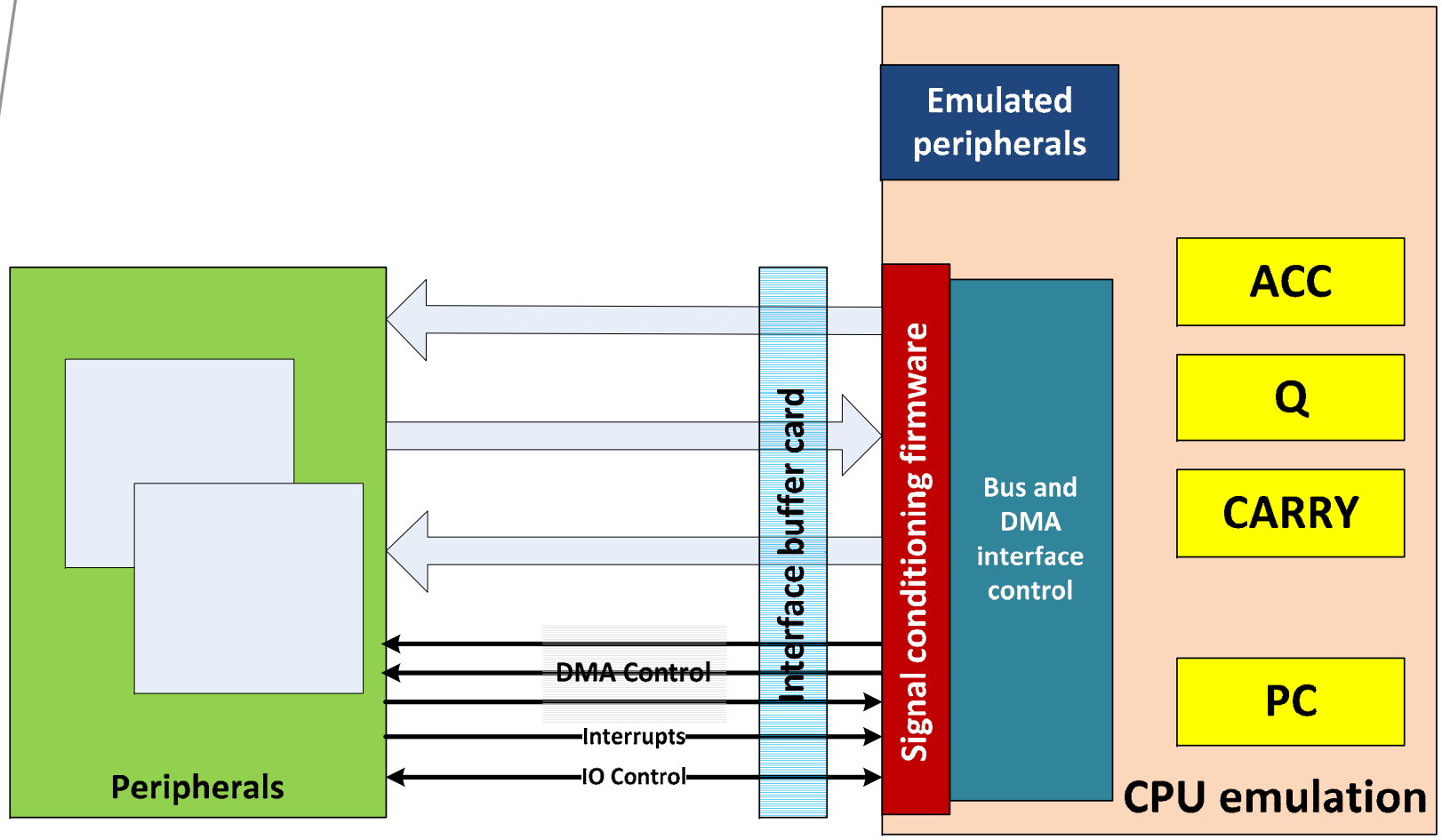


Implementation – hardware and firmware

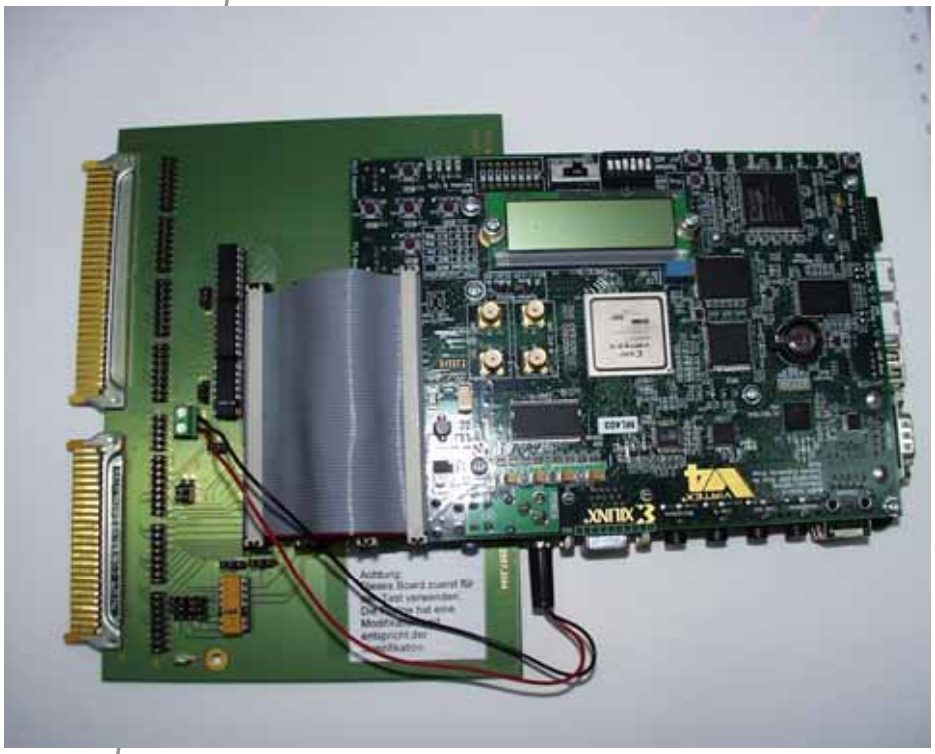
- FPGA “design” using base-system builder from the Xilinx EDK tools
  - Processor clocked at 200MHz (cf. usual 300MHz) to simplify macro instruction timing
  - RS232 (for VDU output)
  - Two-states for busses and control signals etc.
  - ...largely an out-of-the-box solution
- Simple interface conversion/buffer card between FPGA IO and controller busses
  - Mainly 3.3↔5v conversions + simple signal conditioning (R-C networks, pull-ups etc.).
  - Some simple signal filtering in the firmware design
- Two PECs (FPGA and interface) replaced ~15 processor and peripheral related PECs

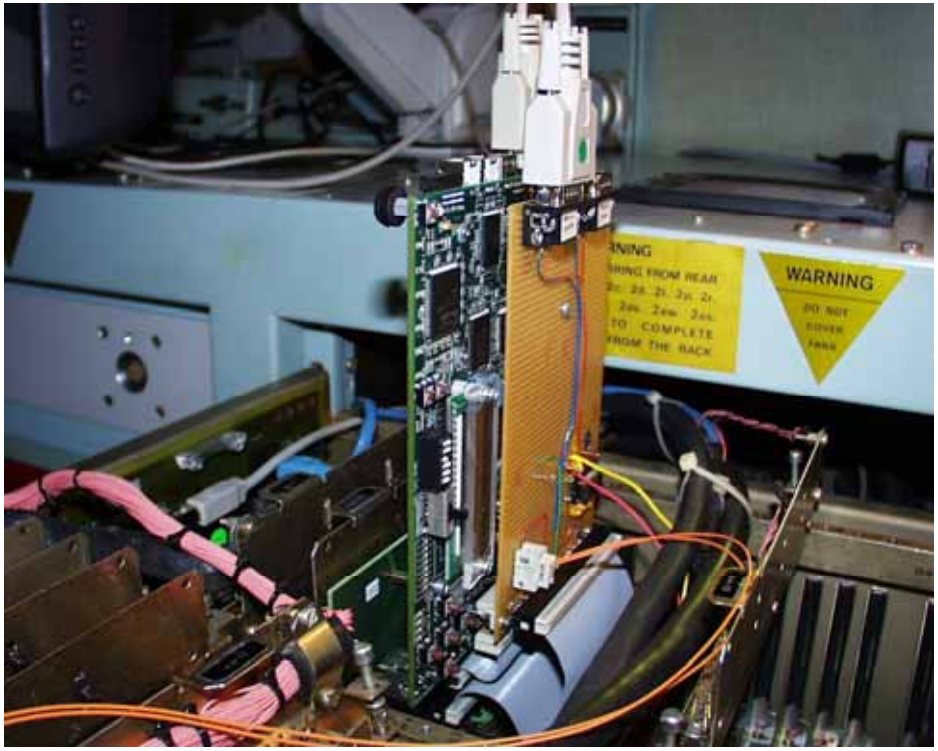
MBDA MISSILE SYSTEMS

Implementation – architecture

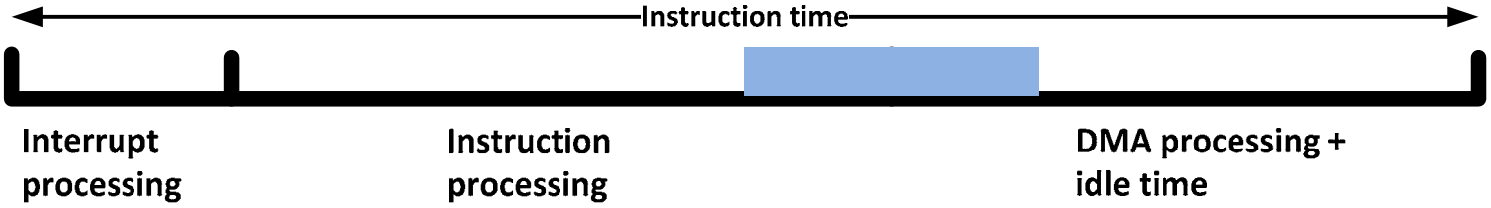


Implementation – installed hardware





Implementation – main processing timeline and code

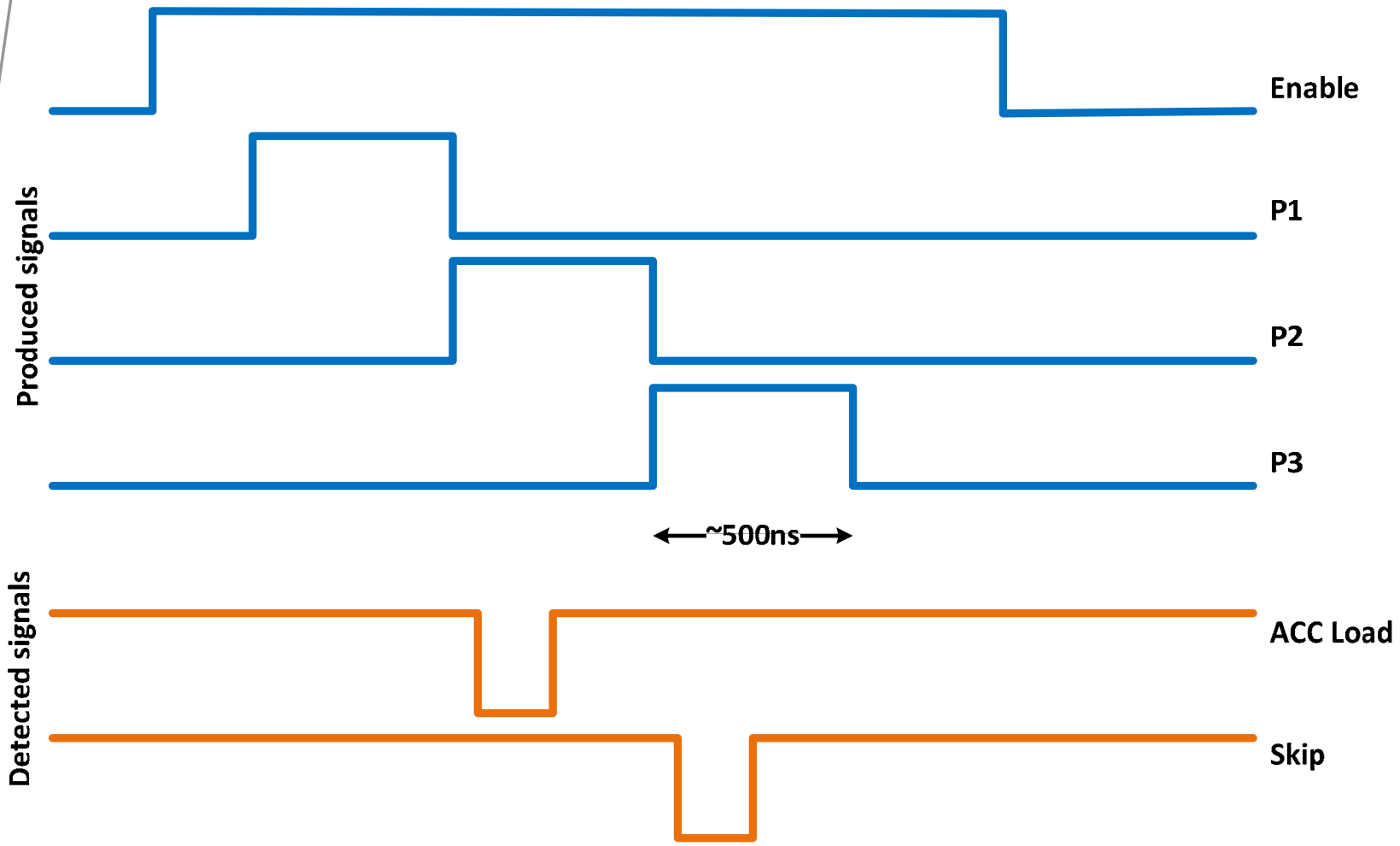


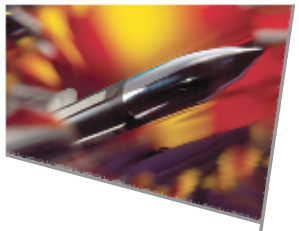

```
loop
   --  Check for any interrupt sources
   --
   Check_for_Interrupts;

   --  Decode and execute instruction
   --
   case Instruction_Type (Memory (PC)) is
      when Addressing =>
         Action_Addressing (Memory (PC));
      when Non_Addressing =>
         Action_Non_Addressing (Memory (PC));
      when IO_instruction =>
         Action_IO (memory (PC));
   end case;

   --  Wait for end of instruction time
   --
   while Instruction_Time_not_Expired loop
      Action_DMA;
   end loop;
end loop;
```

Implementation – software generated IO control signals

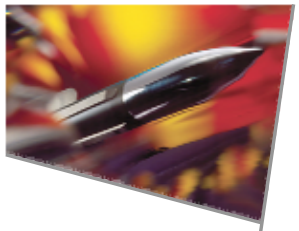




Results

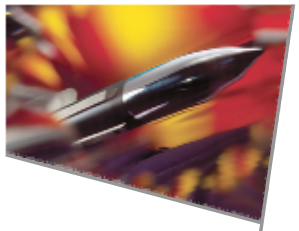
- Fulfilled all original goals, demonstrated the viability of the approach
  - Will be installed in the equipment over the remained of this year (albeit with a hardware update)
- Timing margins appear ample (especially given the reduction in processor speed)
  - Generally 50-75% of available time for any instruction is unused – plenty of time to handle DMA successfully
- … seen as a viable approach to the management of obsolescence
  - This work has ensured that test equipment that was in large part developed in the 1960s will be able to be used into the 2020s

MBDA MISSILE SYSTEMS



Conclusions

- Demonstrates the use of a virtual machine to solve an obsolescence problem
  - Albeit on a relatively simple example
  - … more complex ones have been considered and would be viable
- The key is choosing the right level at which to apply the solution
  - In this case there is no desire to develop the control software any further, if this had been an issue the approach taken might have been different.
  - Here the value was in the ATPs and control software; the approach taken minimised the risk to these
    - As too would have the micro-code approach – but this did not help with peripheral obsolescence (and the micro-code is a generally less well understood)



Conclusions

- The use of Ada provides relatively high-level abstractions to describe the machine
  - Fast development, easily understood
    - Actually would have been a good way to define a model of the processor – some of the textual description was “difficult”
  - “Contracts” helped to define invariants relating to registers
    - Assertion pragmas rather than aspects…
  - Perhaps the use of an Ada → Hardware (VHDL) compiler might allow this technique to be used with simpler (potentially cheaper) FPGA technologies
    - But it is doubtful that this would have been as quick or simple
    - … and given the “production” volumes the hardware costs are not significant

MBDA MISSILE SYSTEMS

QUESTIONS…