

Hardware Independence Using GEDAE

Authors: A. Wilshire, M. Stevens, D. Dominy



- What is Hardware Independence?
- Why is Hardware Independence desirable?
- How do we achieve it?
- Practical Example: Re-porting an existing design
- Conclusions



hardware independent

Definition

adj. ~ Able to operate on many different types of equipment.

Notes

Hardware independent does not imply the ability to run on all equipment. Hardware independent software may require an intermediary program or translation to enable it to run on different platforms; for example, a program written in C must be compiled using different tools to run on different computers.

Richard Pearce-Moses,
Society of American Archivists



“The ability to perform an algorithm on a subset of hardware
efficiently, with minimal effort and with predictable results”





Customers are now interested in “Whole Life Costs”:

- Efficient use of hardware and software
- Low maintenance / repair cost
- Obsolescence contingencies

Hardware Independence can change the development lifecycle:

- Initial Design
 - Delayed selection of hardware
 - Take advantage of the latest generation of hardware
- Technology Insertion
 - Reduced cost and timescales



“Hardware independence is achieved through the judicious specification of standards which promote portability and interoperability of software products”, FAS.org

- Rapid Prototyping & Development tools
 - Virtual Machine
 - Abstraction of Algorithm and Hardware
 - Optimised Libraries
 - Efficient use of the hardware





- To demonstrate hardware independence
 - Take an existing design optimised to run in real-time on embedded hardware
 - Re-port onto alternate hardware
 - Verify performance
 - Assess time to achieve this task



	Original	New
Rack	VME	VME
Host Processor	Solaris	None
Processing Cards	2x Mercury Quad PowerPC	2x Dy4 Quad PowerPC
Disk-array Interface	VMetro RxMDR	VMetro OpenMDR
Switched Fabric	Raceway	Star-Fabric
Operating System	MCOS	VxWorks
DSP Library	SAL	IXLibs

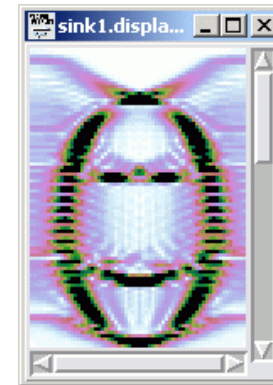
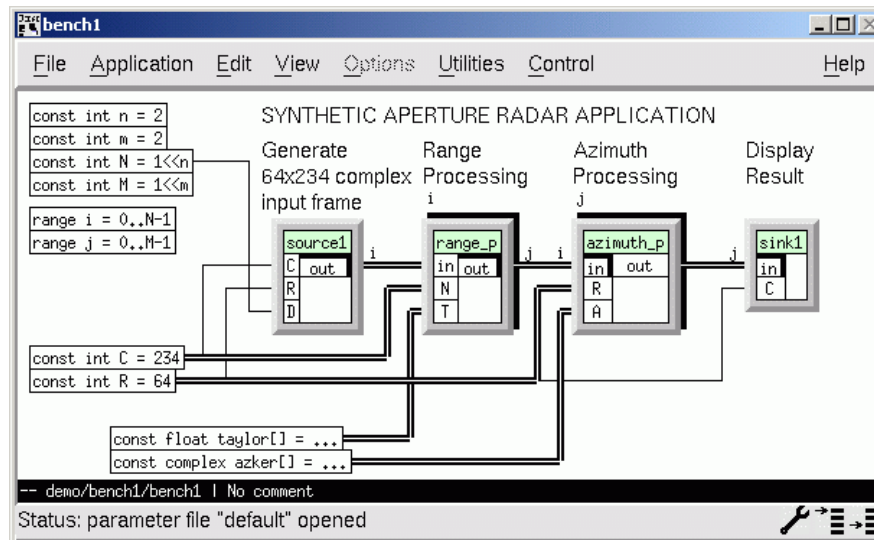


1. Set-up processing boards and BSP
2. Validation tests
3. Identify and revise hardware dependent code
4. Port to new hardware
5. Assess Results



- Gedae Installation has improved
 - Still not fully automated
 - Required manual untar of Dy4 files
 - Required modification of makefile for old version of Exceed
- BSP setup
 - Requires a large number of packages installed
 - CHAMPtools v1.4, StarLink PMC drivers, IPC library, IXLibs, Tornado 2.2, VxWorks 5.5, Cygwin
 - Embedded Configuration File
 - Needed extra information about the transfer mechanisms

- Simple tests to prove operation on target hardware
- Verify data transfers between
 - Host-Embedded
 - Embedded-Host
 - Embedded-Embedded (within a board)
 - Embedded-Embedded (between boards)





- Hardware specific code identified in one primitive
 - “ReadRadarData” Primitive
 - Reads data from the disk-array
 - Interface card and drivers changed requiring different function call.
 - Primitive already designed to allow data to be read from the host or disk-array (using `#ifdef EMBEDDED`)
 - Simple modification to use new drivers



- Select Embedded Configuration
- Modify transfer mechanisms
 - No standardised transfer mechanisms
 - DSA transfer is `dsa_dx` or `dsa_bulk`
 - Stream transfer is `stream_dma` or `stream_msg`
 - Shortcut:
 - Edit the group settings file directly
- Mapping
 - Alter Processor Mapping (if necessary)



- Out of Memory: Can't initialise the processors
 - Default configuration set-up for 32MB memory
 - Needs to be modified to access full 128MB available
- Out of Memory: Can't allocate 1.6GB Memory!
 - First test transferred data from the host processor
 - Using standard fopen command allocates a buffer to transfer the whole data file (1.6GB)
 - Switch to use embFOpen
- Partial Success
 - Consistently processed and displayed first batch of data
 - But intermittent lock-ups afterwards



- Dialogue with Dy4 indicated known problem with IPC
 - Update to use latest versions of BSP software
- Problem now identified as location of comms buffers
 - DSA buffers must be located in lower 64MB of RAM
- Workaround
 - Memory allocated from top
 - Allocate buffer to occupy upper half of memory
- Now Successfully read and process data from the host



- RxMDR disk-array interface card was supplied with drivers that supported raceway transfers
- New OpenMDR supplied with example program
- Extra task to write a high level driver to support transfers from the disk-array over Star-Link



- Once BSP set-up and IPC problems resolved:
- Simple task to run on alternate hardware
- No optimisation was necessary to retain real-time
- New processing card use 25% quicker processors
 - Only part of this was realised giving a 10-15% improvement
 - Some interleaved complex still used – not supported in the optimised signal processing library
- Overall effort 2½ weeks (spread over 1½ months)



- Majority of time was spent setting up the BSP
 - This was a new BSP
 - Future targeting to this BSP should be quicker
- Actual Re-ported was very quick and easy
- Important to keep hardware specific code separate from primitives
- Need to assess level of support in the signal processing library when switching hardware

Note:

Deliberate decision to use similar boards

- Avoids a lot of effort re-partitioning and verify this is support by the algorithm



Any Questions?

