

The Use of Legacy Code in GEDAE

Marcus Alphey

BAE SYSTEMS Avionics

Sensor Systems Division

Crewe Toll House, Ferry Road

Edinburgh, EH5 2XS, UK

marcus.alphey@baesystems.com

Presentation Overview

- Introduction
- Legacy code considerations
- Portability
- Building legacy support into GEDAE
- Use of legacy code in GEDAE boxes
- Modelling considerations
- Testing
- Use on CAPTOR Tranche 2
- Future work
- Conclusions

Introduction

- Reasons for using legacy code
 - Avoids cost of re-development
 - Stable code base for key building blocks
 - Reduced effort to reach required functionality
- Use of legacy code in GEDAE
 - Requires a C-based API
 - Compile and link against external source code or libraries
 - Call legacy functions from within primitive boxes
 - 'Wrapper' to convert data to / from legacy format

Legacy Code Considerations

- Coding standards
 - Conflicts between legacy and GEDAE coding conventions
- Global data
 - Controlling access for multiple processors
- Program constants and macro definitions
 - Relevance of hardware / software references
- Datatypes and data structures
 - Access to fields or elements, conversion support
- Functions
 - Function prototypes
 - Global data
 - Memory allocation

Portability

- GEDAE design ethos
- Isolate hardware-specific functionality
- BSP / e_ functions
- Memory addressing
- Byte ordering and alignment
 - Input and output data
 - Memory images

The image shows two overlapping windows. The top window is the 'Embeddable Box Editor' with the class name 'users/marcus/legacy/receive_data'. The code in the editor is as follows:

```

Include: {
#include "marcus/legacy/interface.h"
#include <e_vmov,h>
}

Init: {
setup_complete = 0;
}

Reset: {
count = 0;
ready = 0;
if (setup_complete != 1) {
setup_complete = e_configure_receiver(Buffer_Address, Buffer_Size);
}
}

Apply: {
int i; /* Granularity loop counter */
int status; /* Status of the receiver */

for (i = 0; i < count; i++) {
/*
Copy the
and conver
*/
e_unpack_da
/* Keep tra
Status: saved fi
  
```

The bottom window is a terminal titled 'skye1:/home/ecr90/users/systems/marcus'. It shows the execution of two hex dump commands:

```

[skye1@marcus>hex -N 128 NT_mem0:0000000
0000000 5400 0000 5500 0000 5600 0000 5700 0000
0000010 5800 0000 5900 0000 5a00 0000 5b00 0000
0000020 5c00 0000 5d00 0000 5e00 0000 5f00 0000
0000030 7000 0000 7100 0000 7200 0000 7300 0000
0000040 7400 0000 7500 0000 7600 0000 7700 0000
0000050 7800 0000 7900 0000 8d00 0000 7a00 0000
0000060 8e00 0000 7b00 0000 8f00 0000 7c00 0000
0000070 9000 0000 7d00 0000 9100 0000 7e00 0000
0000080
[skye1@marcus>hex -N 128 Solaris_mem0:0000000
0000000 0000 0064 0000 0065 0000 0066 0000 0067
0000010 0000 0068 0000 0069 0000 006a 0000 006b
0000020 0000 006c 0000 006d 0000 006e 0000 006f
0000030 0000 0070 0000 0071 0000 0072 0000 0073
0000040 0000 0074 0000 0075 0000 0076 0000 0077
0000050 0000 0078 0000 0079 0000 008d 0000 007a
0000060 0000 008e 0000 007b 0000 008f 0000 007c
0000070 0000 0090 0000 007d 0000 0091 0000 007e
0000080
[skye1@marcus>
  
```

Building Legacy Support into GEDAE

- Compiling the GEDAE executable
 - makeGEDAE
 - Personal_Obj_List
- Compilation flags
 - Conditional processing
 - Optimisation and debugging
 - Include and link paths
- Source and include files
 - Directory structure
 - Archiving issues
- Libraries
 - Forcing linkage

```

makeGEDAE - d:/gedae_4_0/nt/
Print File Edit Options Buffers Tools Help
#!/usr/local/bin/perl
#
# makeGEDAE
#
$ENV{"HOST_ARCH"} = "nt";           # Host system
$ENV{"MAKE"} = "gmake";           # Microsoft's make
$ENV{"PERSONAL_FILE"} = "Personal_Obj_List"; # Personal function box list
$ENV{"CC"} = "cl";                # ANSI C compiler
$ENV{"CFLAGS"} = "-Z7 -D_COMPLEX_DEFINED -DWIN32 -nologo -D_NT_ -DLEGACY";
-DNO_STATIC -DDEBUG"; # Compiler options
$ENV{"PRELINKFLAGS"} = "";        # Not required
$ENV{"CC_LINKER"} = "link";        # Host linker
$ENV{"EXE_SUFFIX"} = ".exe";
$ENV{"XINC"} = "-ID:\Exceed\jdk\include";
$ENV{"XLIB"} = "-defaultlib:HCLXm -defaultlib:HCLXt -defaultlib:Xlib -d";
defaultlib:xlibcon";
--\-- makeGEDAE Mon Feb 24 11:22AM (Perl CVS:1.5)--L1--CO--Top-----
EMBEDDABLE FILES (only needed if you want
# to avoid the make system check and compilation
# when loading a graph)
#-----
PERSONAL_EMBED_FILES = \

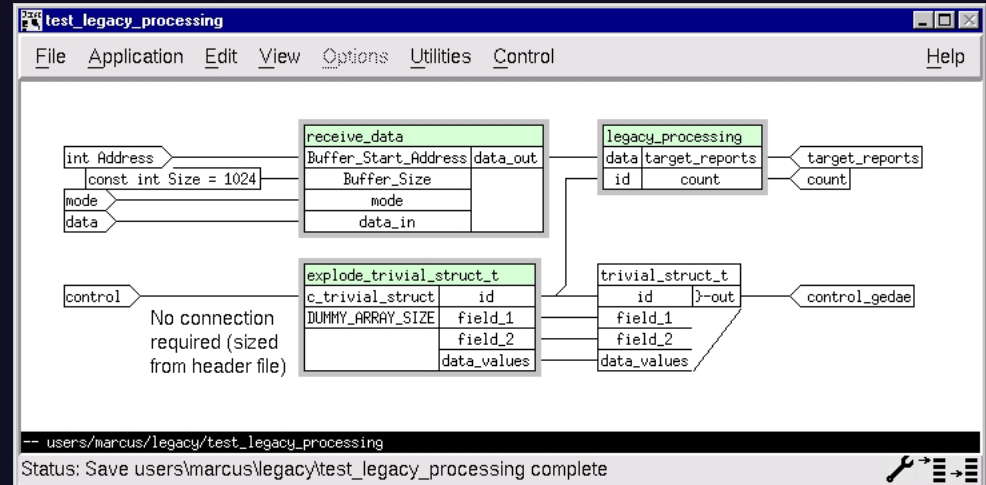
#-----
# ALL USER DEFINED FILES
#-----
LEGACY = interface.o setup.o comm.o
CONVERSION = convert.o misc.o
PERSONAL_HOST_FILES = $(PERSONAL_EMBED_FILES) \
$(LEGACY) \
$(CONVERSION)

#-----
# EXTRA LIBRARIES NEEDED BY PERSONAL FUNCTIONS
#-----
LEGACY_LIB = -L../legacy/libs/legacy.lib
PERSONAL_LIBS = $(LEGACY_LIB)
--\-- Personal_Obj_List Mon Feb 24 11:22AM (Fundamental CVS:1.4)--L13--CO--Bot-----

```

Use of Legacy Code in GEDAE Boxes

- Flowgraphs
 - Local data declarations
 - Typedefs
 - GEDAE
 - C
- Primitives
 - Header files
 - Wrapper code
 - Data conversion
 - Granularity
 - Set-up tasks



Use of Legacy Code in GEDAE Boxes

- Flowgraphs
 - Local data declarations
 - Typedefs
 - GEDAE
 - C
- Primitives
 - Header files
 - Wrapper code
 - Data conversion
 - Granularity
 - Set-up tasks

The screenshot shows the 'Embeddable Box Editor' window with the following C code:

```

#include "marcus/legacy/legacy.h"
#include "marcus/legacy/types.h"
#include <e_vmov.h>
#include <e_vfill.h>
}

Reset: {
  if (setup == 0) {
    initialise_weights(weights);
    setup = 1;
  }
}

Apply: {
  int i; /* Granularity loop counter */
  float fixed_size[Max_Length];
  int16 legacy_id;
  int16 legacy_count;

  for (i = 0; i < granularity; i++) {

    /* Convert the input variable vector to a fixed size one */
    e_vmov(data, 1, fixed_size, 1, *n);
    e_vfill(0,0, &(fixed_size[*n]), 1, (Max_Length - (*n)));
#ifdef DEBUG
    embPrintFloat(Max_Length, fixed_size);
#endif

    /* Cast id to the legacy type */
    legacy_id = (int16)(*id);
#ifdef DEBUG
    printf("legacy id = %d\n", legacy_id);
#endif

    /* Carry out the legacy processing */
    window_transform(fixed_size, weights, legacy_id,
                    target_reports, &(legacy_count));
    *count = (int)legacy_count;
#ifdef DEBUG
    print_report(target_reports, legacy_count);
#endif
  }
}

```

Status: loaded file Text: unchanged

Modelling Considerations

- Dataflow sizing
 - Variable-sized tokens
 - Over-specified variable matrices
 - Enumerated Scheduling
 - Exclusion
- Parameter evaluation
 - Host or target
- Processor loading
 - Distribution of legacy code blocks

Testing

- Differences in use of legacy code
 - Compilation variations
- Debugging
 - Conditionally processed diagnostic messages
 - Check data conversion at inputs and outputs
 - Progress reports
- Comparison with original behaviour
 - Validate key components
 - Consider platform differences
 - Word size, precision
 - Accumulated errors

Use on CAPTOR Tranche 2

- CAPTOR
 - Multi-mode nose-radar for Eurofighter Typhoon
- Tranche 2
 - Re-development to support porting to COTS processor
- Re-use specific legacy code components from Tranche 1
 - Written in C, targetted at single processor, portable
- Re-implement remainder in GEDAE
 - Support functions for data conversion
 - Emulate memory model for some components
- Progressing well

Future Work

- Pre-processing of legacy code
 - High-level code-optimisers
 - Vectorisation
 - Parallelisation
- Improvements to GEDAE
 - Support for legacy source code debugging of primitives
 - Dependency checking for primitives
 - Identify related source and include files
 - Flexibility in organising source and include directories
 - Separate directories for different projects
 - Simplify organisation and archiving

Conclusions

- Use of legacy code in GEDAE primitives can be a powerful way of exploiting existing work products
 - Saves time and effort of re-development
 - External C source code and libraries can be linked into GEDAE
- Legacy coding standards, portability of code and modelling constraints must all be considered
- Legacy function calls made from within GEDAE primitives
 - May require wrapper to convert data to / from legacy format
 - Must also consider granularity and set-up issues
- Scope exists for further legacy support in GEDAE
- Legacy C code successfully used on CAPTOR Tranche 2
 - Savings of time and effort achieved

Questions?