# Software Verification of a Virtual Development Environment for Embedded Software

FEBIANSYAH HIDAYAT, HADIPURNAWAN SATRIA, JIN B. KWON
Department of Computer Science and Engineering
Sun Moon University
Tangjeong, Asan, Chungnam, 336-708
REPUBLIC OF KOREA
havban@gmail.com, hadi198@yahoo.com, jbkwon@sunmoon.ac.kr

*Abstract:* - Virtualization helps developer optimizing potential of their resource. Physical hardware can be represented as software programs in multi-purpose computer. SID is known as framework providing hardware virtualization based on components. Virtual Development Environment for Embedded Software (VDEES) wrapped it inside Eclipse Platform, providing GUI for managing component interactions and creation of custom components. Using VDEES, developers will be helped to write software on top of new virtual components without waiting for the physical components hardware being implemented. It cuts the time, cost and risk to the lowest. Furthermore, VDEES is open source, that way we can enhance and modify as needed.

*Key-Words:* - Embedded Software, Hardware Simulation, SID, Virtual Component

## 1 Introduction

VDEES has been developed since 2005, based on open source platforms. It is build on top of Eclipse platform and uses SID as the virtual environment. SID is component based; it means that to build a new kind of hardware is to build a new component in the SID framework.

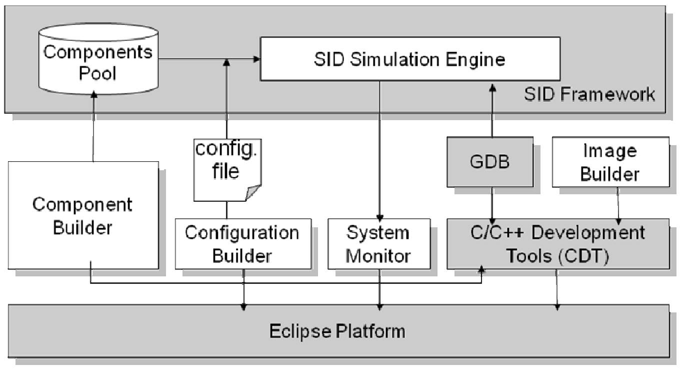


Fig 1. VDEES Architecture

This paper will explain verification steps we have done using VDEES. There are five target programs used for testing. Description about the physical environment and virtual environment will be presented on section 2 and 3. The development and testing of components and binary image building will be included in section 4 to 6. Fig.

Our final goal is the binary image will be able to run on the physical environment (board) and on the SID framework. The binary image itself may be compiled using various compilers as long it is targeted for the specified hardware.

We divided our work into three phases, Physical, Pre-Development, and Development phases. The Physical phase is when the program is able to run on physical environment. We were already on that phase when this project started. The Physical phase will be our comparison with the result of this project. The Pre-Development phase is when we analyze which custom components are needed to be build, and we also decided how detail our work will be on the component. The Development phase realizes our needs from the previous phase. After all the three phases, we can evaluate the correctness and performance using five test programs running on simulated environment..

## 2 Physical Environment

We used an MBA-2410 board as target board that consists of ARM920T with 32 MB of RAM. The board is equipped with LCD screen, touch screen, and keypad. It has some modes to run as options, using Smart Card or using internal memory. In this project, the program is running initially using internal memory, addressed at 0x30000000 since the physical RAM is actually located at that address.

ARM920T has the same instruction set as the ARM7TDMI processor that is supported by the SID framework. The difference is only on the availability of Memory Management Unit (MMU) sub component that is needed when an Operating System (OS) is running on the board. In our case, the MMU is not needed since only one single program, non OS-based, is running. In that case, existing ARM7TDMI virtual component is sufficient.

There are many other details of components that we would skip in the implementation to avoid complexity and only focus on functionality. The cache and buffering in processor and video data would be skipped; instead direct access to memory address will be used.

To communicate among components, the registers are used with determined address. Changing the value of a register will change the behavior of component corresponding to the register. The definitions of components address are listed in the header file of source code.

The main IDE used to develop the program is Code Warrior (CW). It supports many kinds of boards. Debugging feature is also available using AXD debugger from CW. CW is recommended by Atmel Inc. and distributed along with the board with the manual compliant. Another application used is Spider debugger, to do debugging.

## 3 Virtual Environment

We are using VDEES that has SID framework inside it. The VDEES runs on Eclipse with C/C++ Development Toolkit (CDT) plug in support. SID framework should also be installed in the system, it is already included in the VDEES installation package, for more information, you can access this link [7].

VDEES provides four *plugins*, they are *Configuration Builder, Component Builder, System Monitor*, and *Image Builder*. Those components are built on top of Eclipse Platform. As we can see in the Fig. 1, Image Builder and Component Builder are extensions of CDT in Eclipse.

GNU ARM compiler is needed, and is a separated element from Eclipse, but included in the VDEES installation package. Different from regular *gcc*, the *arm-elf-gcc* (GNU C compiler for ARM) will build the binary to comply with the target machine that may have different instruction set than the host machine. The language itself is C as we will explain later on the source code adaptation about some differences between CW's and GNU ARM's syntax and keywords.

*arm-elf-gdb* (GNU Debugger for ARM) is used to do monitoring and debugging of the running application. The two applications communicate through socket with defined port. Then, the value will be displayed in the Eclipse window. VDEES can do monitoring to the level of virtual component variable status. It should give an in-depth view of current running program on the virtual environment.

Unix environment is required to run the SID, since the source code of it is only compatible with Unix platform. Fedora Linux 4 and UBuntu Linux 8.4 are the OS platforms we used in this development, although Cygwin for windows may be used also to mimic the Unix environment.

## 4 Pre-Development

At this phase, we analyzed the pre-existing components of SID, for example, the ARM CPU, Loader, LCD, Timer, Interrupt component. From that list, we can compare it to the real board components which are used in the test program and decide which need to be built.

We listed the required components of the binary program by analyzing the source code. We did not want to implement non existing component in the SID framework while it is not needed in the program. We just need to make sure the program will run well on the virtual environment, by getting the list of registers that will experience value changes on runtime. The registers implementation is important, since the MBA-2410 is accessing components through memory addresses. This gives many conveniences to the developers as they only need to refer a component using regular address. Which register address is accessed shows which component being used.

The list of components we needed to build were *Analog to Digital Converter* (ADC - Handle conversion data from analog to digital data) , *Touch Screen Interface* (using mouse click as event trigger), *LCD Controller* (Process the data from the video buffer and update the the LCD screen), *Timer* (periodically trigger interrupt), *Clock and Power management* (manage the clock speed given to various components), and *Interrupt Controller* (Provide interface to the interrupt queue in the processor). Some of those components are extension of pre-existing one.

We also avoided implementing components that their registers accessed from the test program, yet the behavior of the program will not be affected in term of functionality, for example the speed and status indicator. LED component can be ignored as it only shows debugging performed in the board. Various clock setting can be skipped and only provide single fastest clock, hoping the simulation will run as smooth as possible.

## 5 Development

We move on to the development phase, where we implement components, compile the test program source code using arm-elf-gcc, and create configuration files consisting of components loaded and their connections to each other. The whole processes are done within the VDEES IDE features mentioned before.

## 5.1 Custom Components

VDEES provides wizard to create a custom component, generating basic template files required to build the component. Those template files can be modified to our needs.

C++ editor and builder are available in VDEES as the extension of CDT *plugin* from eclipse. For Tcl/Tk code editing, we use external editor, gedit or vim.

The first component implemented was ADC Controller and Touch Screen Interface using C++ and Tk. The last one implemented is Interrupt Controller, since at first we could actually bypass the code that uses the interrupt component.

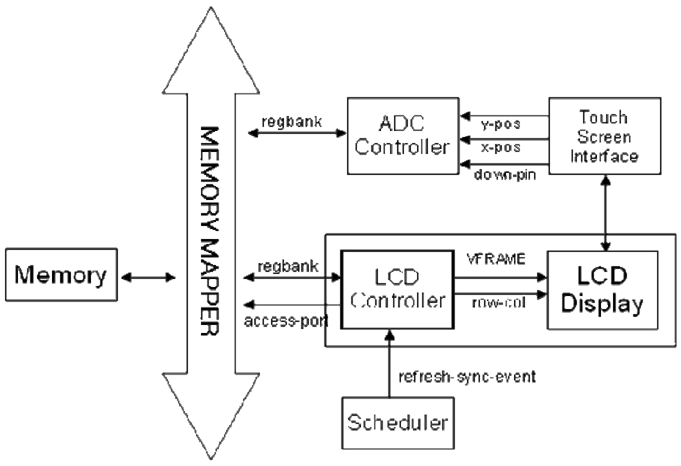


Fig. 2 SID Component Interaction

Some details of hardware are not implemented to provide faster simulation running and simpler implementation. As we can see in Fig. 2, the LCD controller directly accesses the memory from bus, instead of buffering data first in a video buffer as in the physical board. For another example is registers implemented in ADC Controller are not all needed, in this case, we only implement four registers that actually accessed from the test program, they are handling the touch event.

## 5.2 Image Binary Building

As we mentioned before, the physical environment uses CW to build the binary image of program. CW uses almost similar syntax of C and Assembly compared to the GNU ARM compiler. Some keywords are different, so we modified it referring to the GNU ARM code style.

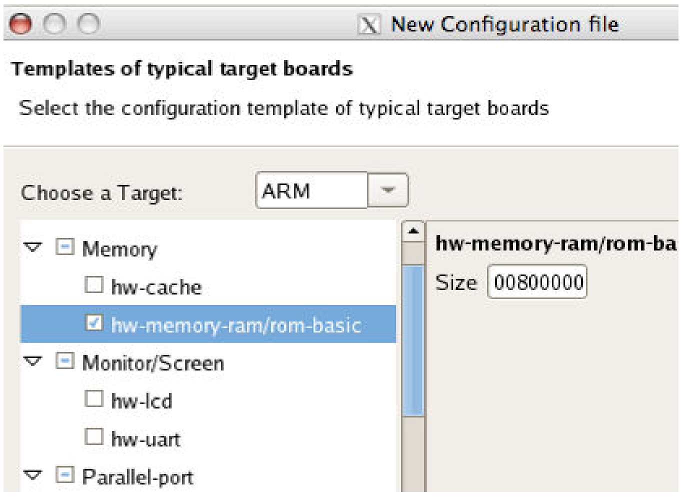
The basic change is modifying the names to be all lower case, since Unix is case sensitive. In the C source code, we changed the interrupt keyword of Interrupt Request Queue (IRQ) from "__irq" into "__attribute__((interrupt("IRQ"))))". And most of other changes were done in the Assembly code where we modified it refers to [6].

We removed code blocks for MMU functions handling. The functions are still not implemented in the existing virtual ARM component. We also do not need the MMU itself since the test programs use none of its functions (no OS).

Some parameters given while linking should be considered. We must put "–nostartfiles" to indicate that we are using different startup files than the host machine. "-Wl,-Ttext=0x30000000" indicates that the code is located at 0x30000000, where the physical memory available in the board.

## 5.3 Configuration File

In order to be able to run the simulation, configuration files should be created, defining which components are loaded and connected and also how they interact. VDEES provides wizard where we can choose from existing components of SID and the custom components we have created.


New Configuration file

**Templates of typical target boards**

Select the configuration template of typical target boards

Choose a Target: ARM

- Memory
  - ☐ hw-cache
  - ☑ hw-memory-ram/rom-basic
- Monitor/Screen
  - ☐ hw-lcd
  - ☐ hw-uart
- Parallel-port

**hw-memory-ram/rom-ba**

Size 00800000

Fig. 3 Configuration File Wizard

After creation of configuration file from the wizards, we can still modify and adapt it based on our needs. We related components manually, since relate is not supported yet in the wizard. Some direct attribute modifications were performed too while debugging. At this current state, we can just run the file using command "sid <file name>.conf" from the command line and validate the configuration.

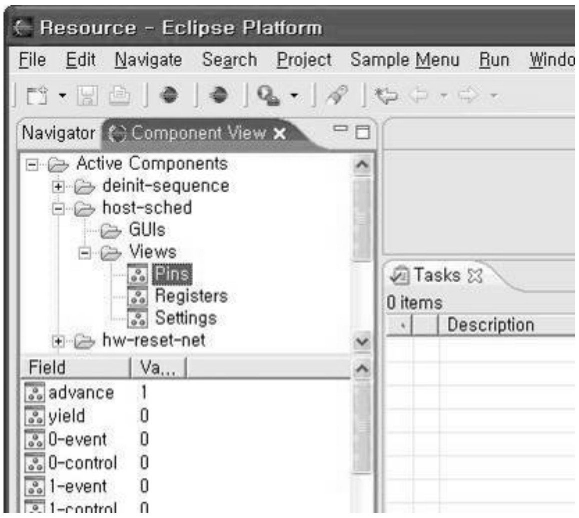
### 5.4 System Monitor and Testing

To run the simulation inside VDEES, we have to setup the simulation environment. This is done by setting port number and target address that GDB will be used. The binary image file and source codes should be provided accessible from the system. This instruction is available in the VDEES manual.

In our experiment, the debugging feature failed to run. It once successfully running in a simple code provided from another case, yet in a more complex program, the GDB can only be commanded to start and stop. Pause and step-by-step code executions are not available.

The problem could lie on the binary program that includes incompatible debugging information. For this suspicion, we've tried various types of debugging format. However, the GDB still refuse to run properly. We assume the problem may lie on the VDEES parameter given to run the GDB. This is out of our reach to fix and we haven't found the good composition also using manual GDB calling from console.

Inability of GDB to run properly implies on monitoring feature of components becomes malfunctioning. We could not view the window of components' properties. Fig. 4 shows how the window should look like.



| Field | Va... |
|---|---|
| advance | 1 |
| yield | 0 |
| 0-event | 0 |
| 0-control | 0 |
| 1-event | 0 |
| 1-control | 0 |

Fig. 4 System Monitoring

## 6 Performance evaluation

Because of problem with GDB, we tested the component using print out debugging manually. We check the validity through analyzing each output in the console, and so for speed execution calculation.

From five test programs, only one program fails to run well on the simulation. The program crashed after showing initial screen and moving for a few steps. Meanwhile, all the four program runs as it should be. However, all of them ran very slow. For each touch input require few seconds for evaluation and 1 second for display refresh. It is far from real-time execution we expected. The scheduling factor in SID should be investigated as it may give unfair execution share among components. We found out also that the components are executing sequentially, it will be better if we could make it parallel. This will probably need modification in the SID source code.

Between printout through stdio for debugging and executing current instruction set in the test program, there is so much lag time. This shows how bad the scheduling of component execution is done.

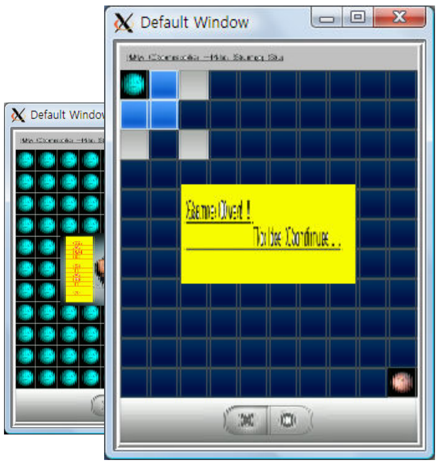


Fig. 5 Running Program in Simulation

## 7 Conclusion

We have developed custom components and verified how VDEES runs. It helps developer to do faster design and testing of newly hardware without waiting long for real implementation of hardware gets available. Debugging feature was still not functioning well, and it will be our future work to investigate. But overall, using VDEES will benefit developer to do simulation before implementing the real hardware.

*References:*

[1] Seal, David, ARM Architecture Reference Manual, Addison-Wesley, 2001.

[2] H. Satria, B. Wibowo, J.B. Kwon, J.B. Lee, Y.S. Hwang, A Virtual Development Environment for Embedded Software using Open Source Software, IEEE Trans. on Consumer Electronics, May 2009.

[3] MBA-2410 User manual, [24 November 2003]

[4] VDEES Homepage, http://cslab.sunmoon.ac.kr/vdees/ , 2009

[5] Ronetix, ARM cross development with GNU Toolchain and Eclipse version 1.1, May 2007

[6] ARM Architecture Reference Manual , 2005

[7] SID reference, http://sourceware.org/sid, 2009