A Top-Down Design Methodology Using Virtual Platforms for Concept Development

Mohit Shah[1], Brian Mears[2], Chaitali Chakrabarti[1], Andreas Spanias[1]
[1]SenSIP Center, School of ECEE, Arizona State University, Tempe, AZ USA
[2]Intel Corporation, Chandler, AZ USA
[1]E-mail: mohit.shah@asu.edu, chaitali@asu.edu, spanias@asu.edu
[2]E-mail: brian.r.mears@intel.com

Abstract

Virtual platforms are widely used for system-level modeling, design and simulation. In this paper, we propose a virtual platform-based, top-down, system-level design methodology for developing and testing hardware/software right from the concept level and even before the architecture is finalized. The methodology is based on using tools such as QEMU, SystemC and TLM2.0 that starts with a functional, high-level description of the system and gradually refines the intricate architectural details. We present our results by testing a novel concept aimed at performing audio blogging. The system under consideration involves the design of a low-power wearable audio recorder, an Android application for user interface and a server for audio analysis. A virtual system consisting of three instances of QEMU and other tools was created to demonstrate the concept and to test this approach. Finally, we describe a suite of tools useful for quickly validating concepts and creating virtual platforms for early hardware/software codesign.

Keywords

QEMU, SystemC, TLM2.0, Android, Virtual Platforms

1. Introduction

Virtual Platforms allow software to be tested prior to silicon availability which reduces Time-To-Market (TTM) for a new product launch. Several products such as Synopsys Platform Architect [1], Bochs [2] and QEMU [3] are used for this purpose. A virtual platform is typically created after the product architecture is fairly stable, which means limited time to use it prior to silicon availability. This design flow is better explained in Figure 1. In this paper, we put forth the idea of starting virtual platform development right from the time when product concepts are being formed and architecture details are not fully available. This allows the product ideas to be explored before key decisions are made, and enables this first virtual platform to gradually be used for product development. This can be seen in the apparent left-shift of the design flow in Figure 2, reducing the TTM even more.

SystemC/TLM2.0 or C/C++ based virtual platforms have been used extensively for modeling embedded devices [4-6]. The authors in [6] modeled an MPEG-decoder using a modified version of QEMU, known as QEMU-SystemC. QEMU, along with QEMU-SystemC, provides a flexible and easy-to-use environment for instantiating virtual models of various devices and processors. Currently, the main focus has been on constructing virtual platforms for modeling a single device such as an ASIC or an embedded mobile platform [6, 7]. However, most of the present applications include interactions between multiple devices. For example, an application based on video tracking involves a sensor for video capture, a server for video analysis and communication between the two devices for complete operation. In addition, the design constraints and objectives of one device might affect the performance of the other. So, a joint optimization needs to be performed across all devices to ensure better system performance.

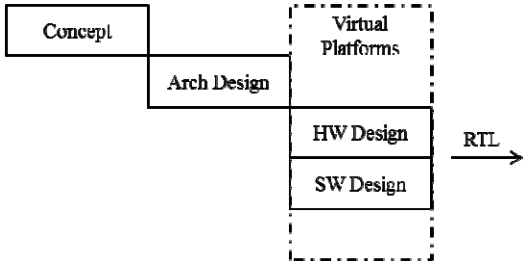


Figure 1: Current design flow

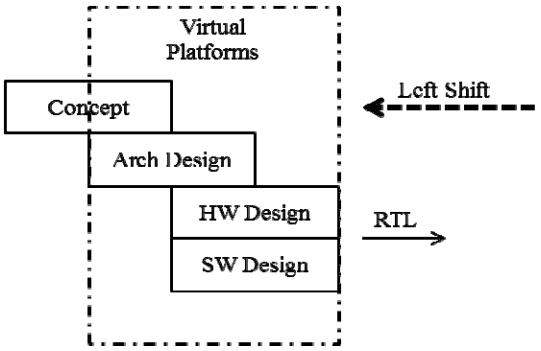


Figure 2: Future design flow

We focus on a top-down design methodology starting with the description of an application related to audio-based blogging or lifelogging. The system is broken down into a set of components or devices such as sensors, mobile devices and servers. Modeled by virtual platforms, each device is initially just an abstraction at the functional level. Then, we identify the constraints and trade-offs that affect the overall system performance. For instance, optimizing the sensor design for ultra-low power consumption, the mobile application interface or tuning the performance of data analysis algorithms running on a server. Taking these issues into consideration, we gradually build a more refined system. Thus, starting from just a concept, we develop a

virtual platform-based model of the system prior to the availability of silicon or architecture details. The same model can be refined further and used during production, thereby reducing the TTM significantly.

Our major contributions in this work are two-fold. We propose a novel top-down design methodology that starts from the application concept at a higher abstraction level and gradually iterates towards a refined hardware and software architecture. Secondly, we describe a set of tools that would allow a designer to build such large-scale applications relying on multiple devices and their interactions. The advantages of using this virtual platform-based methodology compared to simulating devices at the cycle-accurate level are supported by our results explained in section 5. We have also developed a generic framework, called Concept Development Kit (CDK). The CDK is useful for quickly validating and prototyping concepts and for early software development.

2. Design tools

The design of devices such as sensors or smartphones generally involves a main processor for generic tasks and a set of peripheral coprocessors to handle the computationally intensive tasks. The main processor runs an operating system such as GNU/Linux or Android while device drivers are written to access the peripheral coprocessors. QEMU [3] is an open-source virtualization and emulation tool. It can emulate entire systems based on x86, ARM, SPARC and other platforms along with their peripheral devices. It also allows for designers to write their own virtual devices, plug them in QEMU and evaluate their performance. Due to these features, we have chosen QEMU as the basic building block for creating virtual platforms.

SystemC and TLM2.0 [8] are two widely used industry standards for modeling hardware devices and communication interfaces in complex systems. They are favored due to the multiple levels of abstraction they offer. They allow designers to model abstractions ranging from a functional-level and blocking transport mechanism to a register-accurate, cycle-approximate and non-blocking mechanism. Although it is possible to model x86 and ARM instruction sets, the computational overhead of SystemC tremendously slows down the simulation (relative to QEMU). A variation of QEMU called QEMU-SystemC [6] is capable of running virtual hardware devices written in SystemC/TLM2.0 within QEMU. This combination enables high speed CPU modeling with SystemC accuracy for peripheral models.

The architecture of QEMU-SystemC is shown in Figure 3. At the base level, a SystemC link acts as a bridge attached to the PCI, AMBA or IO bus of the platform. The designer's device is attached to this SystemC link. A read or write instruction to the virtual device is first transferred to the bridge, which further translates this instruction into a TLM2.0 compliant instruction and forwards it to the virtual device. This infrastructure is quite useful since it retains the fast execution speeds of QEMU as well as the design abstraction levels offered by SystemC/TLM2.0. We use this version of QEMU specifically for sensor design, since it involves the design of custom peripheral blocks and testing their performance within a complex system.

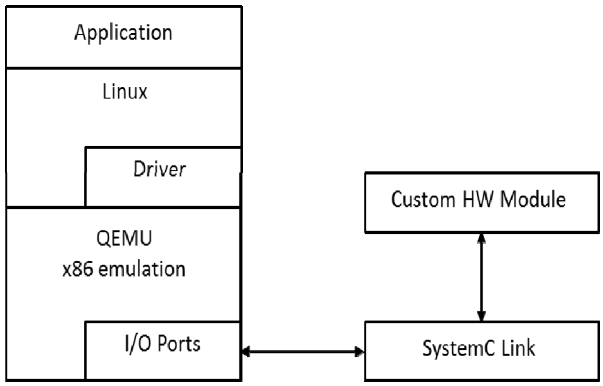


Figure 3: QEMU-SystemC architecture

Similarly, the Android emulator [9] is another stable variation of QEMU capable of emulating a full-fledged Android smartphone. The three variants, QEMU, QEMU-SystemC and Android emulator have been used in this research work to create virtual platforms.

3. Application concept

To evaluate the usefulness and performance of virtual platforms for developing concepts, we have come up with a novel concept of audio blogging or lifelogging [10, 11]. To illustrate this idea and the challenges it poses, consider the following scenario. A user equipped with a wearable audio recorder [12] or using his/her smartphone is able to record audio for one or more days. During this period, the user has a conversation with a friend, sees an interesting police car chase and hears some new piece of music on the radio. At some later point, the user wants to share these events with a friend or colleague or simply wishes to recall what exactly happened. Manually browsing or searching for specific events through a long recording can be a time-consuming task. By employing a range of signal processing and machine learning techniques, we have developed a set of algorithms to automatically segment and annotate long audio recordings. An efficient retrieval mechanism has also been developed to allow users to search through their archives.

To develop an actual product out of this concept, many factors need to be taken into consideration. First, there are constraints on the design of the wearable recording device. It must have ultra-low power consumption to enable continuous recording for 24 to 48 hours. It must be able to compress data while recording so as to minimize the storage requirements. Second, details on the communication of data between a smartphone and the wearable device need to be worked out. For example, a Bluetooth/wireless link between the two devices or a simple wired connection has huge implications on power consumption of the wearable device. Similarly, communication between a server performing audio analysis and a smartphone needs to be studied as well. Hence, it is quite evident that in order to ensure a better

overall system performance, various details and issues need to be addressed. The simplicity of the concept, yet the complexity of its implementation, serves as a good exercise to assess the credibility of virtual platforms in modeling such systems.

4. Designing the virtual system

We break down the system into three components to simplify the implementation of the proposed audio blogging application as shown in Figure 4. The components include a wearable recording device or sensor, a smartphone and a server. Each component is assigned a set of specific tasks to be performed and design constraints it must satisfy. The sensor is responsible for continuous audio recording and compression. Likewise, the smartphone is responsible for an intuitive user interface. The server must be able to analyze the uploaded audio data, annotate and archive it automatically. The individual components are modeled and instantiated using the aforementioned variants of QEMU on a single host machine. The host machine provides the interface for communication between these components. The implementation details for each of these components are addressed below.

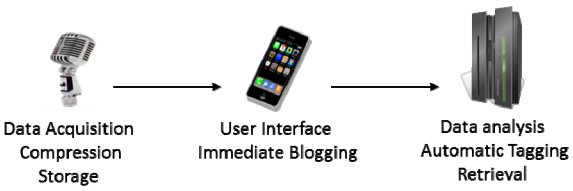


Figure 4: Overview of the virtual system and its components

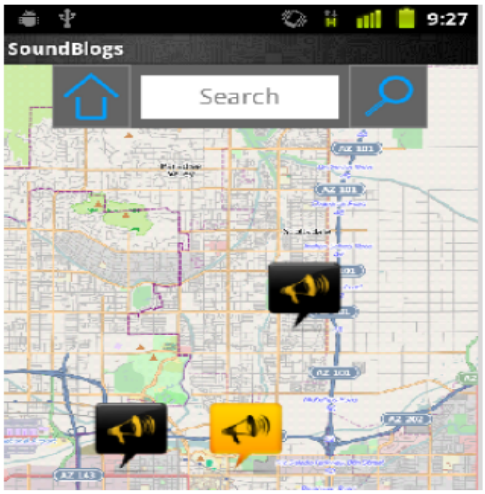
4.1 Sensor model

The sensor is modeled as an x86 system running a GNU/Linux operating system using QEMU. Functionally, the sensor is responsible for compressing the incoming raw audio data and storing it. Here, we have chosen Ogg/Vorbis [13] as the compression algorithm due to its royalty and patent-free nature. Source code for such an encoder is freely available and is run on this virtual device without any modifications and additional coprocessors.

Once compressed, the audio is stored in memory and the sensor awaits further instructions from the smartphone device for data transfer.

4.2 Smartphone model

An Android emulator is used to model a smartphone capable of emulating GPS-based locations and wireless data transfer. Functionally, the smartphone must provide a smooth and intuitive user-interface for the audio blogging application. A user should be able to upload the recorded, compressed and stored audio clip in the sensor to his/her personal archive or publicly blog about it. It should allow the user to search through his/her archives for past



recordings, thereby serving as a useful memory extension.

Figure 5: User-interface on the Android application, *SoundBlogs*

Location plays an important role in categorizing these memories (clips), since sounds are often influenced by the surrounding environment. For example, the audio recorded at a sports event will be significantly different from the audio recorded in a restaurant. Furthermore, by displaying different events using icons or pointers based on their locations on a map facilitates a diary-like approach for audio recordings. An Android user-interface was written for this application, called *SoundBlogs* [14], which runs on an Android emulator and is shown in Figure 5.

4.3 Server model

Again, we use QEMU to emulate a full-fledged GNU/Linux server responsible for serving up web pages as well as performing a set of classification and retrieval algorithms on the user uploaded audio recordings. As is the case with the development of algorithms, they are first written in a high-level language such as Matlab for simulation and testing purposes and then ported to C/C++ for better speed and memory performance during production. To illustrate the ability of QEMU in handling this aspect, we use an open-source counterpart of Matlab, called GNU/Octave [15], on the server model to run these algorithms.

As part of the classification procedure, the algorithm first performs windowing and feature extraction on the audio clip.

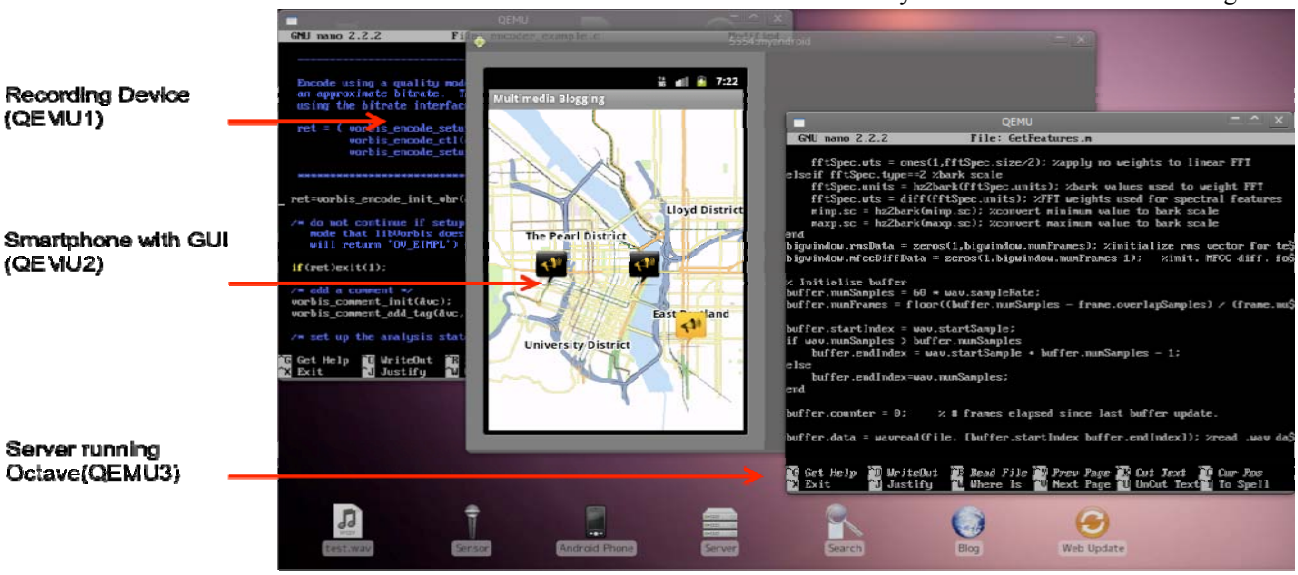


Figure 6: A screenshot of the virtual system

Features such as harmonicity, temporal/spectral sparsity, spectral centroid, loudness and Mel-Frequency Cepstral Coefficients (MFCCs) are used to index the audio clip based on its acoustic contents. Using these features, the algorithm detects the onset and offset of different events in a recording. This segmentation procedure is based on a generalization of Hidden Markov Models (HMMs), known as Dynamic Bayesian Networks (DBNs). By tracking the feature trajectory over time, an event onset or offset can be detected by an abrupt change in the trajectory. Thus, a long piece of recording is broken down into smaller segments or chunks that are easier to analyze.

Each segment is now fed to the annotation algorithm to extract acoustic and semantic tags relevant to that particular sound. The segments are classified and tags are assigned using an HMM-based template matching process. These tags serve to efficiently index the audio data into a huge archive of personal recordings. In order to retrieve specific recordings, the algorithm takes a keyword from the user pertaining to the date, location, acoustic or semantic tags of the desired recording and performs a search on the database to find the most relevant audio clips. More details on the signal processing and machine learning techniques behind these algorithms are discussed in [16].

The virtual system comprising of the three devices instantiated separately on a single host machine is shown in Figure 6. The sequence of actions to perform blogging using this framework can be explained as follows. An audio clip is recorded in to the virtual sensor using an external or the host's in-built microphone. The Ogg/Vorbis encoder program is called to compress the clip and store it. This clip is then simultaneously uploaded to the virtual smartphone and the virtual server. The Android application on the smartphone displays this clip along with ad-hoc GPS coordinates as an icon on the map. The server processes the data and automatically annotates it with relevant tags. Once the user decides to archive/publish this event, all the details, including the audio clip, GPS-based coordinates, date, time, tags, description, are packaged in to a single file and uploaded to the user's blog website or personal archive.

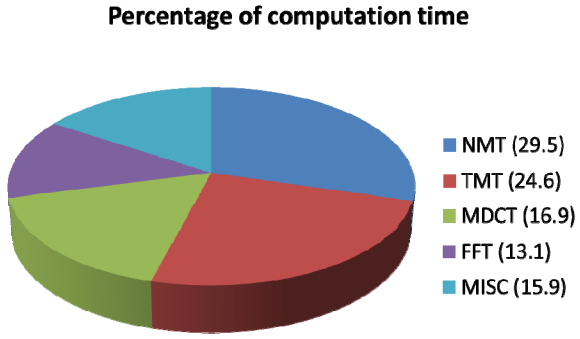


Figure 7: Time profile of the Ogg/Vorbis encoder

5. Refining the system

5.1 Sensor

For the sensor, we are interested in building a non-obstructive, wearable device that can be attached to the lapel. This objective restricts the size of the device and consequently the size of the battery. To meet this criterion, we decide to make use of a stack of two coin cells to power the device. Furthermore, we require that the device should be able to record, compress and store audio for up to 24 hours without having to recharge the batteries. In order to satisfy this requirement, we must ensure that the device operates at ultra-low power. Hence, changes must be made to the compression algorithm to reduce the computational complexity as much as possible.

Before we perform any kind of optimizations on the compression algorithm, we must first identify its computationally intensive routines. A time profile of the

Ogg/Vorbis encoder and the sub-routines is shown in Figure 7. These routines include the Fast Fourier Transform (FFT), Modified Discrete Cosine Transform (MDCT), Tone-Masking Threshold (TMT), Noise-Masking Threshold (NMT) and miscellaneous (MISC) operations related to Huffman encoding and packing the encoded data. Although TMT and NMT constitute 54.1% of the total time, they involve a large number of comparison and threshold operations which are inherently simple to implement and cannot be optimized further. The FFT and MDCT routines, taking 30% of the total time, rely on a significant amount of real and complex multiplication operations.

Our next step is aimed at modifying the FFT and MDCT routines to reduce the overall power consumption. The multiplication operations involved in computing the MDCT and FFT are based on the popular butterfly unit. In [17], Oraintara et al. derive a lifting-based method to convert these units to lattice structures shown in Figure 8. This method allows for quantization and perfect reconstruction without considerable loss of precision. The trigonometric coefficients employed in these computations can be stored as dyadic rational numbers [18]. As a result, multiplications can now be implemented as a series of shift-add operations. We specify the input data and coefficients as 16-bit and 10-bit signed integers respectively. The reconstruction error based on these specifications, as given in [17], is approximately -100 dB. The error is low enough to preserve the quality of recorded audio.

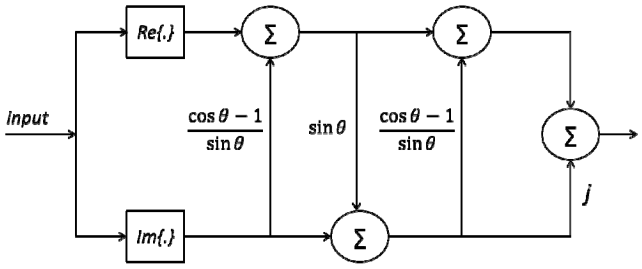


Figure 8: Lattice structure for complex multiplication

We develop virtual hardware devices or coprocessors using SystemC/TLM2.0 to perform MDCT, FFT, TMT and NMT. The MDCT and FFT are implemented using integer-point arithmetic. The TMT and NMT combine to form the Psychoacoustic Model (PAM) and are implemented in mixed integer and floating-point arithmetic. At first, these devices are tested in a pure SystemC/TLM2.0 simulation environment. A block diagram for this approach is shown in Figure 9. A traffic generator sends bursts of data to a generic router. The data is routed to the proper device, either one from FFT, MDCT or PAM, for further computations. A virtual device emulating a ROM is also designed to store sets of coefficients used in FFT and MDCT operations. These blocks are then plugged into QEMU-SystemC through the SystemC interface. The Ogg/Vorbis application code is modified to transfer these operations to the peripheral blocks. For this purpose, special device drivers are written on top of the GNU/Linux operating system. The QEMU-SystemC architecture for this design is shown in Figure 10.

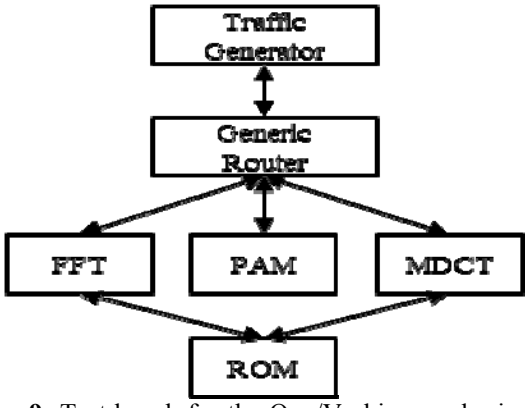


Figure 9: Test-bench for the Ogg/Vorbis encoder in a pure SystemC/TLM2.0 simulation environment

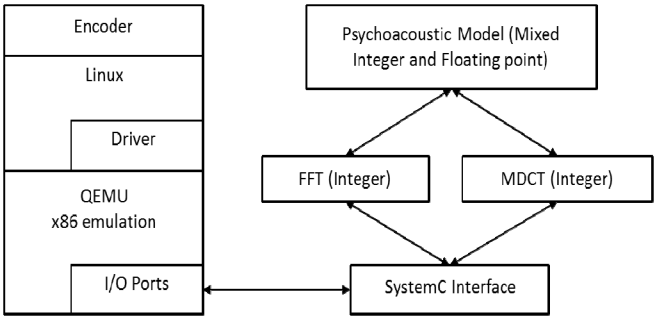


Figure 10: QEMU-SystemC architecture for the refined sensor model

Table 1: Time taken to encode 25 seconds of audio data on different platforms

Platform type (x86-based)	Time (seconds)
Real PC	1.3
QEMU	2.1
QEMU-SystemC	20.4

To evaluate the efficiency of QEMU-SystemC in running complex applications, we tested the Ogg/Vorbis encoder application on three different platforms – a real PC, QEMU and QEMU-SystemC with coprocessors for FFT, MDCT and PAM. The time taken by each platform to encode 25 seconds of audio data at a rate of 64kbps is documented in Table 1. It increases approximately by a factor of 15 on a QEMU-SystemC platform compared to a real PC. This is explained by the additional number of instructions required for the transfer of data between QEMU and the peripheral devices across the SystemC interface. However, the simulation time is considerably low compared to simulating the entire system at cycle-accurate level. Thus, QEMU-SystemC facilitates rapid prototyping and validation at the cost of increasing the level of abstraction.

5.2 Smartphone

We have made use of location-based information in our Android application to enhance the interface. This application is now ported from the QEMU-based Android emulator to an actual smartphone to enable real-time location capture. The application is also tested by various users and their feedback is used to update the interface.

5.3 Server

The server model includes a fully functioning algorithm for automatic annotation and retrieval written in GNU/Octave. However, there is a large overhead in using such simulation tools in terms of speed and memory performance. To improve upon this, we port the algorithms to C++. The modified server model now closely resembles a real-world server. The algorithms are simultaneously deployed on an Amazon EC2 cloud-based server. This allows the actual smartphone application universal data access using its wireless capabilities.

The system is updated with these refined models and the performance is evaluated again. This iterative, top-down design procedure is shown in Figure 11. The projected final system could be an implementation of the sensor on an FPGA device, a user application on a smartphone and the algorithms deployed on an actual server.

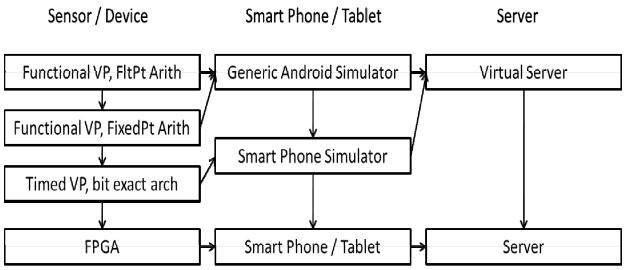


Figure 11: Incremental top-down design methodology

6. Concept Development Kit (CDK)

Although the methods and results have been discussed specifically with respect to audio blogging, the same framework can be used for the development of any application that involves multiple devices in the same configuration i.e. a sensor or an array of sensors, a smartphone/PC and a server. Consider applications such as a medical device for continuously monitoring a user's health or building a gesture recognition controller for mobile devices. The medical device or gesture recognition controllers stated here are essentially data acquisition devices and can be modeled as sensors. Coprocessors for acquiring and storing data can be modeled using SystemC/TLM2.0 and then plugged in to QEMU-SystemC. Similarly, a smartphone or a PC to present results to the user can be modeled using QEMU or the Android emulator. Finally, computationally intensive algorithms for analysis can be deployed on a server modeled using QEMU. Similarly, for a video-based tracking application for surveillance, a video camera acts as the sensor, the control monitors act as the PC-based interface and all the computation is done at the server end. A collection of these devices and the entire application can be modeled using our proposed framework. Hence, using the same tools and framework presented in this paper, we can model a variety of different concepts and applications.

Based on this similarity across different applications, we propose a generic framework, Concept Development Kit (CDK), which would be made available to hardware and software designers alike for rapid prototyping and testing of novel concepts. The CDK includes SystemC and TLM2.0 for the design of custom hardware and QEMU-SystemC to evaluate their performance when integrated into a complex system. It also includes the Android emulator, along with templates and documentation, for designing user applications and interfaces. To model data processing servers with QEMU, the kit includes a number of different tools such as GNU/Octave and Python for deploying algorithms; Linux, Apache, MySQL and PHP (LAMP) for server administration and designing web applications such as blog websites or social media networks. The transfer and communication of data between these different QEMU instances is handled by an easy-to-use program based on the SSH protocol, provided with the kit. Once the concept prototype is tested, the existing virtual platform can be further refined to become the main platform during production.

Figure 12: A system for gesture recognition using our framework

7. Conclusions

We have shown that using QEMU and QEMU-SystemC, we can create virtual platform models to simplify the design and implementation of a novel concept such as audio blogging. We employ a top-down iterative design methodology to develop such concepts, thereby facilitating design in a rapid and parallel fashion. By taking the design objectives and constraints of different devices into consideration, we ensure that the developed application is jointly optimized for better overall system performance. The possibility of refining and using the same virtual platform at a later stage, during production, for example, effectively helps to reduce the TTM significantly. The CDK, a collection of tools and libraries, provides an easy-to-use framework for design and development using this methodology.

Although it is widely used as an emulator, QEMU is intended for functional emulation only and not as an extensive tool for hardware design. For instance, we cannot extract timing details from the QEMU-SystemC environment, such as the exact number of cycles or instructions taken to complete an operation. Our future work will involve studying methods and techniques to lower the

abstraction level in QEMU-SystemC. This involves the study and implementation of methods to include cycle-accurate and register-accurate hardware descriptions and to build synthesizable models starting from just an application concept. Furthermore, future work will also be directed towards implementing different communication protocols presently available in addition to an SSH-based protocol only as part of the CDK. This extension would offer designers and developers to model their application as close as possible to a practical implementation.

8. References

[1] http://www.synopsys.com

[2] http://bochs.sourceforge.net

[3] F. Bellard, "Qemu, a Fast and Portable Dynamic Translator", Usenix Annual Technical Conference, 2005.

[4] A. Dion, E. Boutillon, V. Calmettes, E. Liegon, "A flexible implementation of a Global Navigation Satellite System (GNSS) receiver for on-board satellite navigation", Conference on Design and Architectures for Signal and Image Processing (DASIP), pp.48-53, October 2010.

[5] K. Gruttner, F. Oppenheimer, W. Nebel, F. Colas-Bigey, A. Fouilliart, "SystemC-based Modeling, Seamless Refinement, and Synthesis of a JPEG 2000 Decoder", Design Automation and Test in Europe (DATE), pp.128-133, March 2008.

[6] M. Monton, A. Portero, M. Moreno, B. Martinez, J. Carrabina, "Mixed SW/SystemC SoC Emulation Framework", IEEE International Symposium on Industrial Electronics, June 2007.

[7] S. Abdi, H. Yonghyun, Y. Lochi, C. Hansu, I. Viskic, D. Gajski, "Embedded System Environment: A framework for TLM-based design and prototyping", IEEE International Symposium on Rapid System Prototyping (RSP), pp.1-7, June 2010.

[8] http://www.systemc.org

[9] http://developer.android.com

[10] V. Bush, "As we may think", The Atlantic Monthly, July, 1945.

[11] D.P.W. Ellis and K.S. Lee, "Minimal-impact audio-based personal archives", 1st ACM Workshop Continuous Archival and Retrieval of Personal Experiences, ACM Press, 2004, pp. 39–47.

[12] B. Clarkson, N. Sawhney and A. Pentland, "Auditory context awareness via wearable computing", Perceptual User Interfaces Workshop, 1998.

[13] http://www.xiph.org/vorbis

[14] http://www.soundblogs.org

[15] http://www.gnu.org/software/octave

[16] G. Wichern, J. Xue, H. Thornburg, B. Mechtley, and A. Spanias, "Segmentation, indexing, and retrieval of environmental and natural sounds," IEEE Transactions on Audio, Speech and Language Processing, vol. 18, no. 3, pp. 688–707, 2010.

[17] S. Oraintara, Y. Chen, T. Nguyen, "Integer fast Fourier transform," IEEE Transactions on Signal Processing, vol.50, no.3, pp.607-618, Mar 2002.

[18] T. Tran, "The BinDCT: fast multiplierless approximation of the DCT", IEEE Signal Processing Letters, vol.7, pp.141-145, June 2000.