Embedded Systems©

Analog, Digital, and Microcomputers

By

Raymond L. Barrett, Jr., PhD, PE
CEO, American Research and Development, LLC

1.0 Embedded Systems Introduction

This course traces background history leading to current "Embedded Systems" from the introduction of system classification concepts through the development of digital computer concepts, and first-practice digital computer embedded systems leading up to the current methodologies, tools, and practice involved in the design of Embedded Systems.

The digital computer hardware will be traced from the vacuum-tube era through the advances of solid-state technology to today's integrated circuits. The software will be shown in parallel with introduction of abstract languages, operating systems and "hard" real-time software, program code and library practices, through C-Code editing, compilation, and system building. Hardware Design Language (HDL) concepts will be introduced in the context of adding digital functionality to the included microcomputer capabilities.

The Top-Down design methodology will be introduced in the context of Embedded Systems development using both open-source and proprietary tools from an Engineering System Level (ESL) through an implementation at the demonstration-board level with two different Embedded System Integrated Circuit (IC) examples. The design-flow through the use of tools will be discussed prior to the example use of those tools so that the "big picture" is seen first. A simple Delta- Sigma ($\Delta\Sigma$) analog-to-digital converter will be used as an analog/digital or mixed-signal example with some modeling and simulation relating to the analog functions and some relating to the digital functions. The use of the top-down approach will make the system simulation of behavior clear prior to the "binding" of the functions to analog or digital components.

Two different implementation hardware boards: the BeagleBoard featuring a Texas Instruments (TI) OMAP™ IC with an ARM processor and TI proprietary Digital Signal Processor (DSP) onboard, and Cypress Semiconductor PSoC® 5 with an ARM processor and proprietary configurable analog and digital blocks on a "First Touch" board will be introduced as a hardware target candidates.

Software development tools will be introduced and representative usage shown for both open-source and proprietary tools using a C/C++ path in both an open-source Linux environment and a MicroSoft Windows™ environment. Code generation automation will be discussed and manual code entry discussed in the context of the tools.

The example system will be explored in detail from within the context of the Cypress Semiconductor PSoC® 5 "First Touch" environment with some alternatives introduced. The

steps necessary and the code example for the PSoC® 5 "First Touch" environment will be referenced, as well as the necessary hardware and software requirements to enable the example in the BeagleBoard will be shown and links provided to obtain the accelerometer, display, cables, and software required. The MathWorks PolySpace™ semantic run-time code checker software will be shown with illustrations of the checking capabilities.

2.0 Systems Engineering

A system can be defined as a collection of components with input, output, and possibly memory and processing between. A simple system with input in one domain with translation of a data mode to output in another domain qualifies as a system. A sensor of a physical property with an electrical output is a simple system. The addition of a data processing component permits additional transformations such as scaling, noise reduction, averaging, etc., between the input and output. The addition of memory permits storage and retrieval of prior measurements or retrieval of information such as scaling factors, alarm limits, etc.

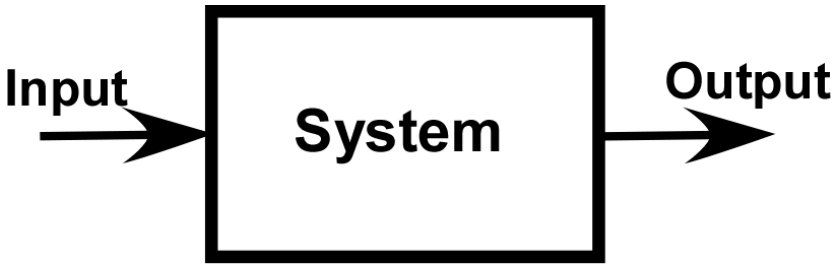

Figure 2.0 – Simple System Block Diagram

There is no general constraint for systems to be constructed in any particular domain and may be electrical, biological, mechanical, fluidic, or other combination. However, modern usage usually refers to "Embedded Systems" as being electronic in nature and having some computational capabilities. We are presenting autonomous electronic systems, although they may have interfaces to the other domains mentioned. Further, we include systems containing digital computing hardware and software, too. Processing is performed by analog circuits and digital components typically comprised of digital logic and memory elements connected to perform arithmetic and logical transformations on the representations of data obtained from the input components and a result directed to the output components.

Before the widespread introduction of modern digital computers instrumentation systems were comprised of analog computing capabilities, but the "Embedded" label began to be applied if a digital computer was a component of the system. To some degree that practice is true today, but with some modification to reflect current integrated circuit technologies. In

some "Embedded Systems" it would be difficult to identify an actual computer as distinct from digital logic alone.

3.0 Embedded Systems Genealogy

Invention of the general-purpose computer was originally a proof of concept for calculating purposes by John Vincent Atanasoff in the 1930's at Iowa State College. The concept was developed into an applied version by John Mauchly and John Presper Eckert for the military and called the Electrical Numerical Integrator and Calculator (ENIAC). The ENIAC was used to calculate ballistic trajectory solutions for artillery-firing tables, an application that would be practical today as an embedded computer application but the ENIAC was far too large at ~30 tons weight, consuming ~160 kilowatts of power, and unreliable because of its containing more that 17,000 vacuum tubes.

Because the vacuum tube was unreliable, other technologies, including the magnetic amplifier (magamp) as perfected in Germany and applied for V2 rocket and battleship fire control applications, were the preferred analog means for what would be recognized today as embedded control systems applications. Following the invention of the point-contact transistor by John Bardeen, Walter Brattain, and William Shockley, and its following technology developments into the bipolar junction transistor by Shockley, the general-purpose digital computer embraced transistor technology. Later, the integrated circuit versions of bipolar "TTL" and "ECL" technologies, N-MOS technology, and today's "CMOS" technology advanced the packaging density and shrank computer sizes..

Much digital computer system improvement was driven by data processing applications in business, but Hughes Aircraft made some use of the emerging digital computers in control in 1954 for automatic pilots, and Ramo-Woolridge (which later became TRW) applied computers for a Texaco oil refining operation in 1959. These applications were embedded digital computer applications for control and either replaced or augmented analog control system components.

There followed a natural progression of embedded digital computing applications from the large "mainframe" systems to minicomputer systems to microcomputer systems, as well as simple digital components used in embedded systems alongside analog counterpart sub-systems. All implementation techniques are still available from large mainframes, through VPX and PCI-Bus card-cage structures that have replaced mini-computers, and into System-on-Chip (SOC) implementations in modern systems. The improved technologies increased capabilities while reducing size and increasing the market breadth. Today, we still have large

systems with mainframe control as well as miniscule systems with one or more digital computers embedded within single integrated circuit (IC) devices.

4.0 Embedded Systems Hardware

Components in a system may have continuously variable signal quantities associated with their values or discrete sets of signal values. Similarly, the values may exist at all instants or time or only at discrete instants when a measurement is made. Those systems with continuously variable magnitude signals available in a continuously variable time fashion are "analog" systems. Those systems with discrete variable magnitude signals available in a discrete-time fashion are "digital" systems. There are two other possible combinations of systems with continuously variable magnitude signals that are only available at discrete-time instants such as Pulse-Amplitude Modulated (PAM) systems, and those with discrete variable magnitude signals that include continuous-time instants such as Pulse-Width Modulated (PWM) systems. The four cases are illustrated in figure 4.0 below.

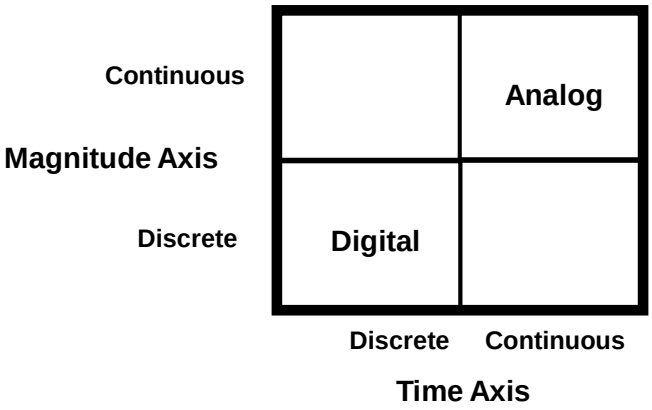

Figure 4.0 Systems Classification Matrix

An "Embedded System" may contain hardware sub-systems that are comprised components from any portion of the matrix of figure 4.0 above. It is common to have some analog components at input and output functions, with an entire digital computer included for its capabilities.

To distinguish a collection of digital components from a digital computer, the digital computer contains at least one Central Processing Unit (CPU) with some form of digital Data Path and Controller that exerts control over the Data Path. The Data Path structure may also contain one or more Arithmetic-Logic Units (ALU) and some form of Memory or Register Array for storing arguments presented to and results taken from the ALU. Generally, there are two forms of memory included, one form of memory for sequencing the Controller

known as Instruction storage, and one form of memory for containing data. It is possible for the instruction memory to be distinct physically from the data memory in "Harvard" architecture, or for both to reside in a common memory structure in "Princeton" architecture. A reduced form of Harvard architecture uses a collection of digital logic to replace the function of instruction memory and the Controller becomes a simple "State Machine" without any possibility of later changing the sequence of operations. In all cases, the purpose of the digital hardware is to convert a digital input signal into a digital output signal.

The same components used in a digital computer structure may comprise simple logic functions converting digital input signals to digital output signals, but without the sequential-state nature of the Controller; sometimes it is difficult to label the collection of logic as a digital computer in an embedded system.

Between analog sub-systems and digital logic subsystems, conversion techniques are employed to turn analog signals into digital signals (Analog-to-Digital, called A2D conversion), or conversely to turn digital signals into analog signals (Digital-to-Analog, called D2A conversion). The digital signals are represented as a simultaneous set of signals with one of two possible discrete values designated as a "one" or "zero," with each individual signal labeled as a Binary-Digit or "bit." The digital signals can exist in the simultaneous "parallel" fashion on a "bus," of bits or be converted to a sequential representation of "serial" single-bits on a one-bit bus. It is common to employ bus widths in "powers of 2," (i.e., 1, 2, 4, 8, 16, 32, 64, etc.), but there are exceptions with 12-bit bus and other structures. Complicated systems have been constructed with specialized dedicated digital signal-processor (DSP) computing hardware that blurs the distinction between digital computer structures and digital logic alone. Such DSP hardware is more likely to be encountered in high-speed signal processing associated with radar and sonar systems that are dedicated to a single purpose computation.

Included within the scope of embedded systems hardware are digital computing systems and devices from existing large systems that used for industrial control, some commercial microprocessor and microcomputer IC component systems, and Intellectual Property (IP) designs owned and licensed for inclusion in "System-on-Chip" (SoC) designs. Using existing, proven, design components in an embedded system can reduce risk and cost associated with achieving system objectives.

The Integrated Circuit (IC) technology that underlies and enables advances in architecture and performance has been driven by the observed/predicted relationship known as "Moore's

Law" describing the interrelationships between the lithographic technology of the IC and other factors. One relationship is the observation/prediction that the linear dimension of a feature decreases enable a doubling of transistor count in every ~18-24 months. That relationship prediction has served as a "roadmap" for the technology developers for over 40 years, with device dimensions shrinking and the 2300 transistor count of the Intel 4004™ processor increasing to over 2 billion transistors in the Intel Quad-Core Itanium Tukwilla™. Along with the transistor-count prediction are "corollaries" that describe the feature size, clock speed, and other factors associated with product manufacturing. The combination of higher speed clocking with increased transistor count has had the effect of exponentially increasing the power dissipation of single-processor products, resulting in a modern data-point of a quad-core design. The clock speed is no longer increasing exponentially, but power is increasing because the area of IC designs is still increasing. With processor multi-core designs enabled in a "System-on-Chip" (SoC) environment, there are "Symmetrical Multi-Processor (SMP)" of 4, 8, and 16 cores, as well as other asymmetrical architecture alternatives available to system designers.

In a later discussion with embedded system examples, we will introduce two systems, each with an Advanced RISC Machines (ARM) 32-bit microcomputer. One example includes a Digital Signal Processor (DSP) capability in addition to the ARM machine, and the other includes the capability of programmable logic and analog functionality in addition to the ARM machine within a single IC.

5.0 Embedded Computer Systems Software

The primary advantage of employing a digital computer within the embedded system is associated with its programmability. Economy of scale permits the construction of a single digital design that performs as many distinct functions as it is programmed to perform. It is the nature of programmability that gives rise to the concept of "soft" in software. Insofar as modern digital logic has enabled programmable digital logic devices, we also include a concept of "firmware" that is midway in nature between hardware and software. Generally, firmware is associated with device-level programming that is unavailable for alteration under routine program control, but much software is employed in that fashion, too.

Insofar as computer systems evolved as a parallel development of hardware technology improvements, the software written to execute on the hardware evolved also. One major innovation in software engineering occurred with the evolution of "Operating System (OS)" software. Essentially all of the functions associated with the support and enabling of a particular hardware implementation were collected into the OS software, making the

programming interface at a higher-level of abstraction than that associated with each hardware collection in a system.

For example, a system containing a disk drive requires the software to enable the disk drive function. Each disk drive manufacturer has different ideas about how their product interfaces to a system and the manufacturer usually writes the necessary software "driver" or "handler." The OS is written so that the rest of the system can utilize many different disk drives by selectively including a particular set of components in a "system configuration." Popular OS software today includes "Windows" from Microsoft, Linux from the "Open-Source" community, and "Solaris" from SUN. In addition, there is a classification of "Real-Time OS (RTOS)" software constructed for some embedded applications that require high-performance allocation of computing resources to ensure timely availability of capabilities. Some examples include VxWorks from Wind River, QNX from QNX Systems, as well as RTLinux from an "Open-Source" community. Not all embedded applications require the performance that an RTOS offers.

In addition to OS software, "Applications Software" is executed by a system. The software is generally written in a 'High-Level" language like "C," or "C++" and a "Compiler" employed to translate into an "Assembly Language" form that is "Relocatable" in the memory. The "Assembly Language" form is specific to the computer hardware and a different compiler or option is required for each hardware target. An "Assembler" is employed to convert the human-readable "Assembly Language" format to the pattern of bits that is the machine language "Object Code" form. For applications that require highest performance, some programming may be done in "Assembly Language," but even the most talented programmer usually avoids any attempt to program in machine-language object code.

High-level "Hardware Description/Design Language (HDL)" tools in VHDL and Verilog languages are available for description and simulation of digital hardware. The code is compiled to a "netlist" of standard digital logic functions from a library of components for construction of digital logic in "Application Specific IC (ASIC)," Field-Programmable Gate-Array (FPGA)," and other forms of logic implementation.

Simulation software is available for analog circuit evaluation such as "Simulation Program for IC Evaluation (SPICE)," digital circuit evaluation such as VHDL and Verilog simulators, and mixed-signal circuit evaluation such as "Advanced Mixed-Signal (AMS)" simulator, and Verilog-A, and the Ptolemy II tool available from the University of California, Berkeley. MathWorks has versions of the popular Matlab™ and Simulink™ software available with

other tools that convert system models to "C" language code, and eventually to executable code for embedded processors.

6.0 Embedded Systems Development

The embedded system can be designed using several methodologies, but the one presented here is based on a "top-down" methodology. A purely top-down approach begins with a simple block diagram such as shown in section 2.0 in figure 2.0 above.

There are a number of methodologies supporting the top-down approach, but all have the common objective of supporting a high level of abstraction for system modeling to capture behavior independent of the implementation issues. The SystemC language, the Ptolemy II tool, Matlab™, SPW™, and Simulink™ are a few tools that support the emerging "Electronic System Level (ESL)" approach to top-down, high-level, system behavior capture and verification. One advantage of the Ptolemy II tool, SPW™, and Simulink™ is that these tools support a design entry process at the block-diagram level that is easily comprehended.

Regardless of the tools, a system specification is usually written with definitions of the system output requirements. Starting from the output definitions ensures that all requirements are met when all output behaviors are satisfied. Next, the available input signal definitions are specified that are required to meet the output dependencies. In the process, a framework is established for the transformations required to obtain the outputs from the available inputs. The signal processing requirements between input and output are defined, beginning with each output and defining the information requirements necessary to provide that output. Usually, the outputs are not a simple scaling or translation of inputs, but require a history of inputs, outputs, and interactions. When all output requirements are met from such transformations, a basic system requirement specification is complete. This is a good milestone for a formal review of the requirements definition.

To the extent that the designer is able, a top-level simulation can be constructed in a design abstraction tool such as the UC-Berkeley tool, Ptolemy II or the Simulink™ tool from MathWorks. Both tools offer the construction of a simulation model to capture system behavior without specific reference to the underlying hardware implementation, deferring the "binding" of variables to the implementation mode until the behavior is captured.

In the case of the heterogeneous computation capabilities of the example systems discussed later, this deferral makes it possible to construct and compare the performance of different bindings. For example, the Beagleboard permits sub-system components to target either its general-purpose ARM™ processor or TI DSP core, and the PSoC5 allows a similar re-target

of function to its general-purpose ARM™ processor or some combination of analog and digital sub-systems. General-purpose ARM™ and DSP processor targets are served by the "C" code-generation capabilities of the tools but FPGA sub-system implementations are better supported by VHDL or Verilog language code generation. The high-level system modeling tools defer the target decision until after behavioral modeling meets the system objectives.

Until recently, there have been few options for translation of the behavioral model from the ESL tool into a form appropriate for the target implementation hardware, but some tools are evolving to ease the translation process. Armed with the correct behavioral model and a target set of hardware, the designer may choose to make the transition manually or using the new translation tools. The automation process supports a "correct-by-construction" methodology for the implementation and verification is eased.

Ptolemy II - The UC-Berkeley tool, Ptolemy II is a GUI-driven modeling framework that supports a growing set of well-defined "Models Of Computation (MOC)" domains. Domains exist for continuous-time modeling of analog behaviors, discrete-time modeling, discrete-event modeling, finite-state machine models, and numerous other domains including wireless system behaviors. Code generation to the open-source EclipseRT tools is supported, as well as VHDL code generation.

We have inserted an illustration from the open-source Ptolemy II environment that describes a Sigma-Delta Modulator sub-system example for control of an accelerometer. It has been chosen because it is amenable to the example target systems described in later sections. The accelerometer output is taken as a sub-system input and represents the response of the spring-mass-damper mechanical response of the accelerometer to a combination of stimuli. There are two sources of stimuli, one may be considered to be the exogenous external acceleration applied, and another the endogenous sub-system output required to maintain mechanical stasis as a zero-error feedback signal from the accelerometer. The example is a mixed-mode simulation with the external accelerometer model in continuous-time as an analog external sub-system and the feedback controller as a digital sub-system within the bounds of the target embedded control system. The choice of defining the endogenous, internal sub-system as a digital component need not be taken as a constraint as we shall see later in consideration of the Cypress PSoC target system.

This particular system example is easily modeled using the Ptolemy II tools and the MathWorks Simulink™ tool equally well. Some modification to the common example will be required and supported as the capabilities of each target embedded system are explored.

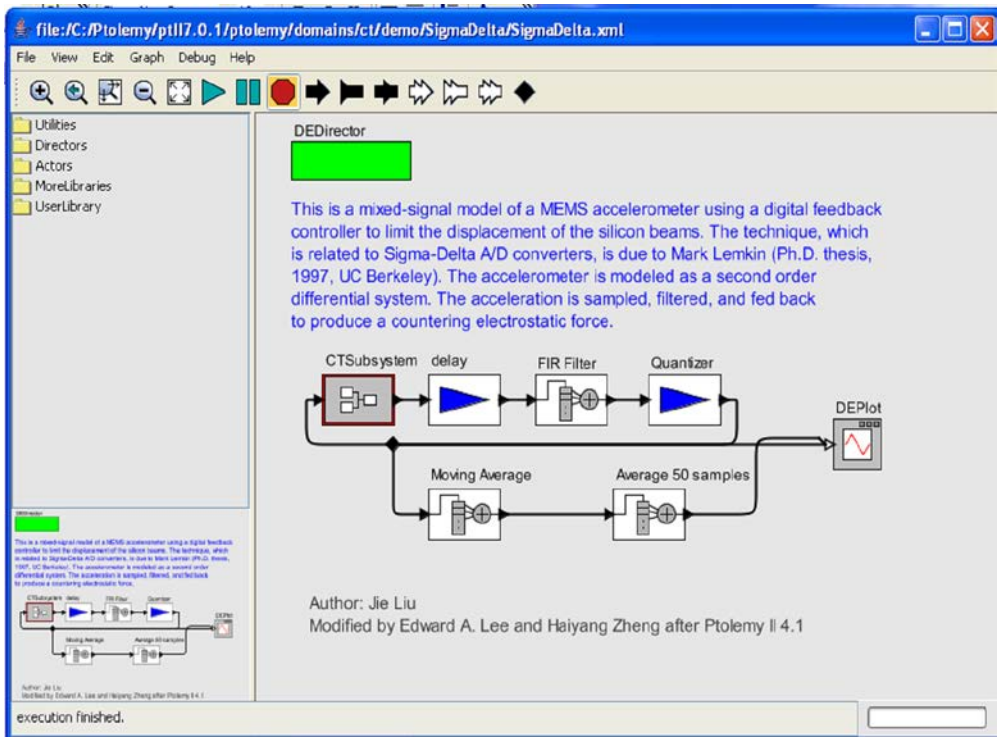

Figure 6.0 The Sigma-Delta Design Example from Ptolemy II Tutorials

A free distribution of the Ptolemy II tools can be obtained from the UC, Berkeley site at the link given below, subject to the copyright notice included at the end of this course.
http://ptolemy.berkeley.edu/ptolemyII/ptII7.0/index.htm

We show the Ptolemy II system diagram obtained by following the tool's links in figure 6.0 above. The page is reached beginning from the start-up page by selecting the Tour of Ptolemy II link to the Heterogeneous Models section and the section/link that follows.

- *Mixed-Signal Modeling*: SigmaDelta (see also Switching Continuous)
  This example shows how to combine continuous-time modeling with discrete-event modeling to get mixed-signal modeling. The example models a MEMS accelerometer where a digital circuit implements feedback control and A/D conversion (a design due to Mark Lemkin).

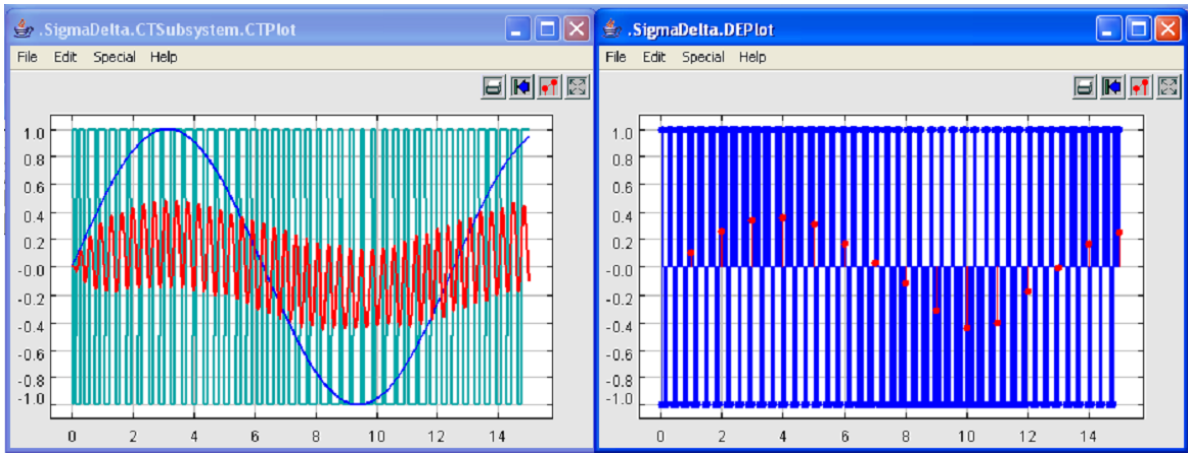

Figure 6.1 The Sigma-Delta Design Results from Ptolemy II Tutorials

The model shown above in figure 6.0 invokes the Vergil tool and is run using the instructions at the top of the "Tour" page. The "Results" shown in figure 6.1 above are obtained from the example. The documentation and tutorials are complete, but take some familiarization with the navigation to/from locales within the framework.

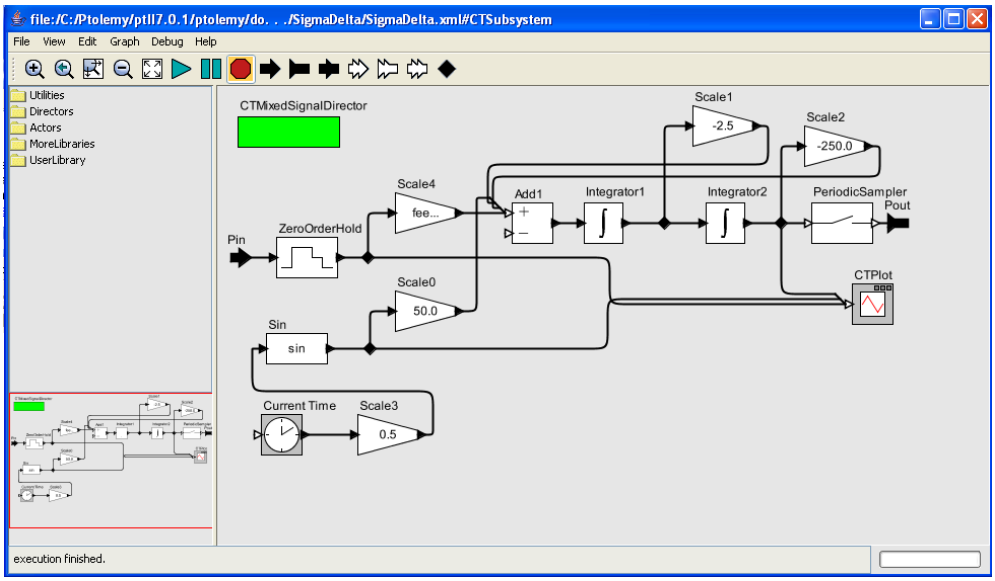

Figure 6.2 The Accelerometer "CTSubsystem" from the Tutorial

The model shown above in figure 6.2 is the Continuous-Time (CT) subsystem representing the double-integrator model of the accelerometer behavior and includes the exogenous sinusoidal acceleration stimulus applied to the device. The results for the CTSubsystem are

plotted in the leftmost panel of figure 6.1 above with the exogenous sinusoidal acceleration appearing as the blue trace, the Pulse-Width Modulated (PWM) input from feedback appearing as the green trace, and the position response appearing as the red trace.

The model in figure 6.2 above represents the kinetics of the accelerometer with the "Add1" block representing the summation of forcing functions. One force results from the exogenous sinusoidal acceleration formed by the "Sin" block driven by a scaled time variable, one from the electrostatic PWM feedback Voltage, one from the viscous damping proportional to the device velocity, and the fourth from the spring suspension restoring force. With Newton's Law: ***F = ma*** solved for the acceleration and the mass constant "***m***" carried implicitly in each of the scaling constants prior to the summation of the "Add1" block, we write describing equations for acceleration, velocity, and position in [6.0], [6.1], and [6.2], below.

$$a_{Accelerometer} = 50a_{Exogenous} + Scale_4 V_{PWM} - 2.5v_{Accelerometer} - 250x_{Accelerometer} \qquad [6.0]$$

$$v_{Accelerometer} = \int a_{Accelerometer} dt \qquad [6.1]$$

$$x_{Accelerometer} = \int v_{Accelerometer} dt \qquad [6.2]$$

The ZeroOrderHold (ZOH) block at the input to the CTSubsystem model accepts the discrete-time signals from the input pin and provides a continuous PWM signal as stimulus to the continuous-time components of the model. The PeriodicSampler block at the output of the CTSubsystem model accepts the continuous-time position signal "***x***" and provides a discrete-time pulse train at the output pin.

Following the Ptolemy II example, we construct a continuous-time subsystem block we call "CTTAccel" within the MathWorks Simulink™ tool as shown in figure 6.3 below. The "CTTAccel" subsystem model is similar to the Ptolemy II example except the ZOH and PeriodicSampler block equivalents are at a higher system level.

Also, we have not constructed the signal-averaging portion of the Ptolemy II example within the Simulink™ version of the example.

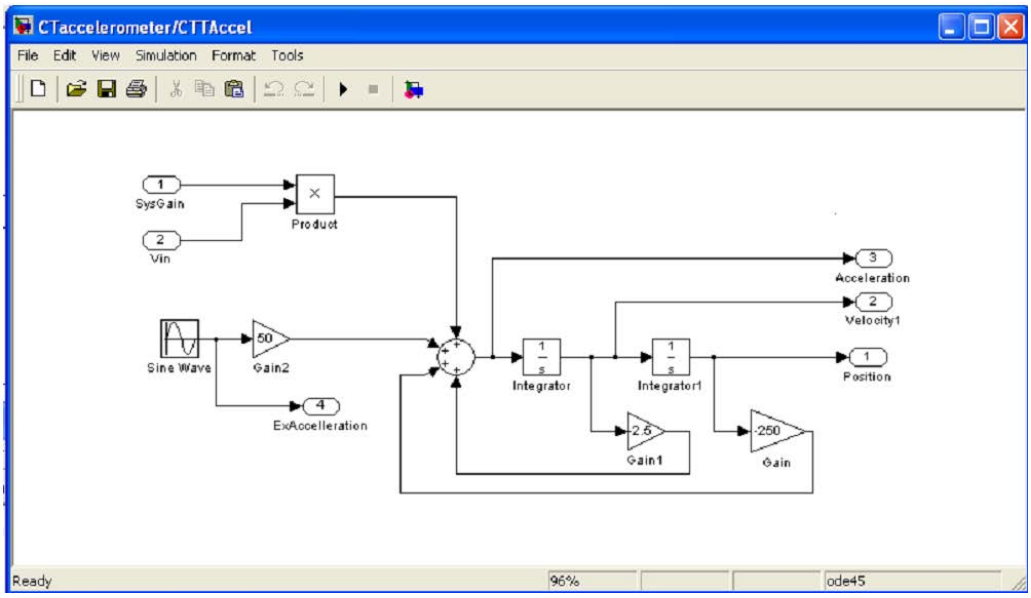

Figure 6.3 An Accelerometer "CTTAccel" Subsystem in Simulink™

Also, the acceleration and velocity state variables are made observable in the Simulink™ model, but only the comparable signals are shown in figure 6.5 below, following the comparable top-level system diagram in figure 6.4 below.

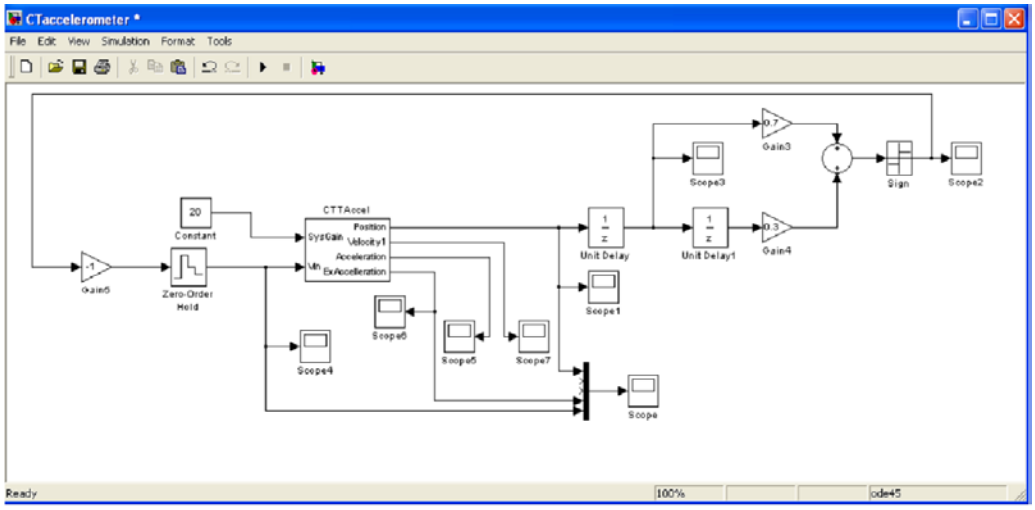

Figure 6.4 "CTAccelerometer" System in Simulink™

For the Simulink™ case, the transformation from continuous-time to discrete-time is achieved using the 1/z delay block with the same delay time used in the Ptolemy II tool.

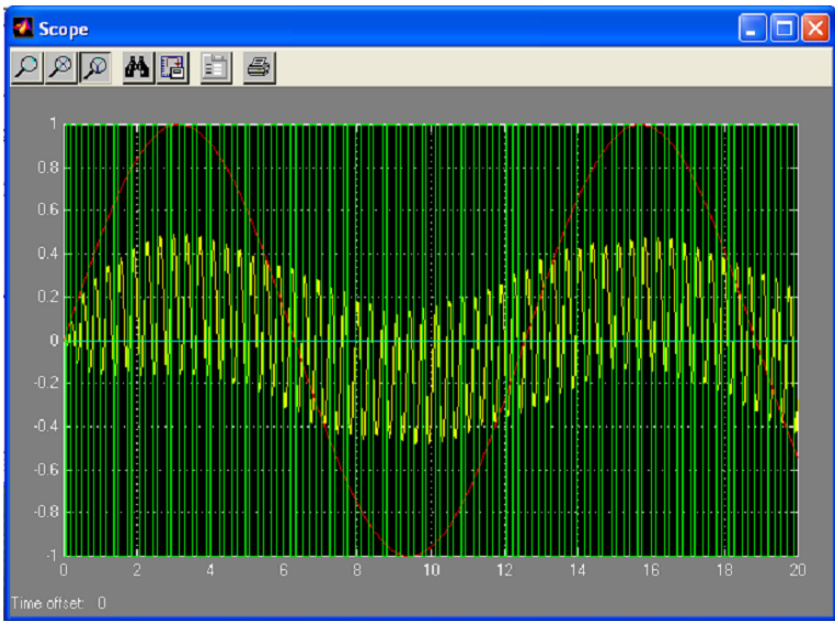

Figure 6.5 Results from Simulink™

The high-level model is incrementally refined into finer and finer sub-system components until the entire input/output and environmental factor behaviors are met. Tradeoff comparisons can be made between MOC alternatives, and code generations for target hardware for each sub-system and a library of functional blocks accumulated for future re-use. For any C/C++ code generated, MathWorks offers the "PolySpace™" tool that checks for run-time errors and code correctness and avoids considerable code semantic verification effort for those languages, regardless of the target hardware.

The code generated by the high-level simulation is compiled/assembled for the target hardware and an emulation environment, such as those discussed in the following sections and employed to verify the behavior in the target hardware.

7.0 Embedded Systems Examples: OMAP ARM/DSP Processor on Beagle Board

The "Beagle-Board™" is a relatively inexpensive (~$149 from DigiKey) experimenter board that is built using a 720 MHertz Texas Instruments (TI) OMAP3530 processor. Recently, a version with Giga-Hertz clock speeds is becoming available. The TI OMAP3530 has an ARM Cortex™-A8 32-bit microprocessor and a TMS320C64x+™ DSP core on the primary processor IC, along with enhanced DMA processor, and POWERVR SGX™ Graphics

Accelerator. Along with the TI processor, the board adds a USB port for communication to a desktop or other computer, a DVI-D port for video, stereo audio outputs, 2 Gbytes of NAND Flash memory, and several other interfaces including to an LCD panel.

The open-source community supports the "Beagle-Board™": http://beagleboard.org/

Other links are available at the website for free software, a free book on "OMAP and DaVinci Software for Dummies©," as well as a number of projects. DigiKey lists the TI OMAP3530™ processor component as a not-stocked item for $69.99, but requires larger volumes than unit quantities, and Arrow Electronics lists versions with unit-quantities available from stock in various versions for $47.13 to $93.66, should the IC be designed into a target embedded system different from the "Beagle-Board™."

The Symbian, Ångström, Ubuntu Linux, and QNX Neutrino RTOS are open-source OS software that can be downloaded for free from the beagleboard.org site. TI supplies a free software distribution without run-time royalty charges, including a DSP/BIOS real-time task scheduler to support the TMS320C64x+™ DSP core on the OMAP3530 processor.

8.0 Embedded Systems Examples: PSoC® 5 ARM/Analog/FPGA Processor

The Programmable System on Chip (PSoC) is a family of embedded processor system IC components manufactured by Cypress Semiconductor. The latest addition to the family is the PSoC® 5 that includes an ARM Cortex™-M3 32-bit microprocessor, memory interfaces, EPROM, DMA Controller and interfaces for CAN 2.0, I2C and USB 2.0 on the IC. In addition to the ARM Cortex™-M3 Processor, the IC has an on-board configurable digital array known as a Universal Digital Block (UDB) comprised of a programmable logic area, datapath, control and status components, a programmable I/O array and an analog subsystem. The analog subsystem has a two-channel filter block, a configurable Delta-Sigma A2D, two 12-bit Successive Approximation A2D converters, four D2A converters, configurable switched-capacitor blocks, configurable filter block, and a collection of comparators and amplifiers. A Field-Programmable interconnection fabric similar in architecture to that used in an FPGA is used to configure analog signal paths. More will be shown about the structures in the context of the design example later. The First Touch board is listed at the Cypress Semiconductor for $49 at the site: https://secure.cypress.com/?id=2218&source=header

9.0 Embedded Systems Tools

Separate tools are generally required for hardware design and software design activities, unless an un-configured system-level product is assumed in the design. For implementation

in Field-Programmable Gate Array (FPGA) and custom Application-Specific Integrated Circuit (ASIC) hardware, considerable additional hardware design work may be required.

Hardware design activities may include an "Application-Specific IC (ASIC)" design of an integrated circuit, a "Printed Wiring Board (PWB)," the "Bill of Materials (BOM)," the "Printed Wiring Assembly (PWA)" including the components from the BOM, and a higher-level mechanical assembly package including the PWA, enclosure, and other components for the entire embedded system target purpose. Sub-assemblies may be obtained at any level for inclusion in the design process.

Translation/Synthesis Tools: To enable translation of ESL simulation block diagram information to a form useful to implementation, a number of tools are available. The Ptolemy II tool is "software lab for experimenting with multiple concurrency formalisms for embedded system design," and has extensive capabilities beyond the modeling and simulation shown thus far. The "objects" shown in the simulation diagrams are called "actors" within Ptolemy II, and carry additional capabilities beyond those shown thus far. Associated with many of the actors is a "codegen helper" capability that generates code for the actor. A specific helper is required for each actor to produce code in a specific language.

Helpers for the Ptolemy II to translate to C/C++ code for statically scheduled domains, including "Synchronous Data Flow (SDF)" and "Heterochronous Dataflow (HDF)" domains, are discussed in the Ptolemy II documentation installed with the software. Additional discussion of the use of "helpers" for VHDL synthesis is contained in the paper "VHDL Code Generation in the Ptolemy II Environment," available for download at: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-140.pdf

The Ptolemy II tool makes reference to the popular "Eclipse" open-source tool for C/C++ code development and references for using the Eclipse tool to access codegen helpers from within Ptolemy. Quite a few helpers are currently available, and there are instructions for constructing helpers that talented users can employ. The codegen "automated" code generation capability is a "work-in-progress" and should not be considered a "bullet-proof" "plug-and-play" option for the novice.

The Eclipse open-source tools can be obtained at: http://www.eclipse.org/
For C/C++ development, the CDT functionality must be included and is available at: http://www.eclipse.org/cdt/downloads.php There is a set of video tutorials available for the eclipse tool at: http://sourceforge.net/projects/eclipsetutorial/files/Eclipse%20Workbench/

There is another "webcast" recording about C/C++ development using CDT (beware that there are audio recording difficulties evident in the presentation) at: https://admin.adobe.acrobat.com/_a300965365/p25766209/?launcher=false&fcsContent=true&pbMode=normal

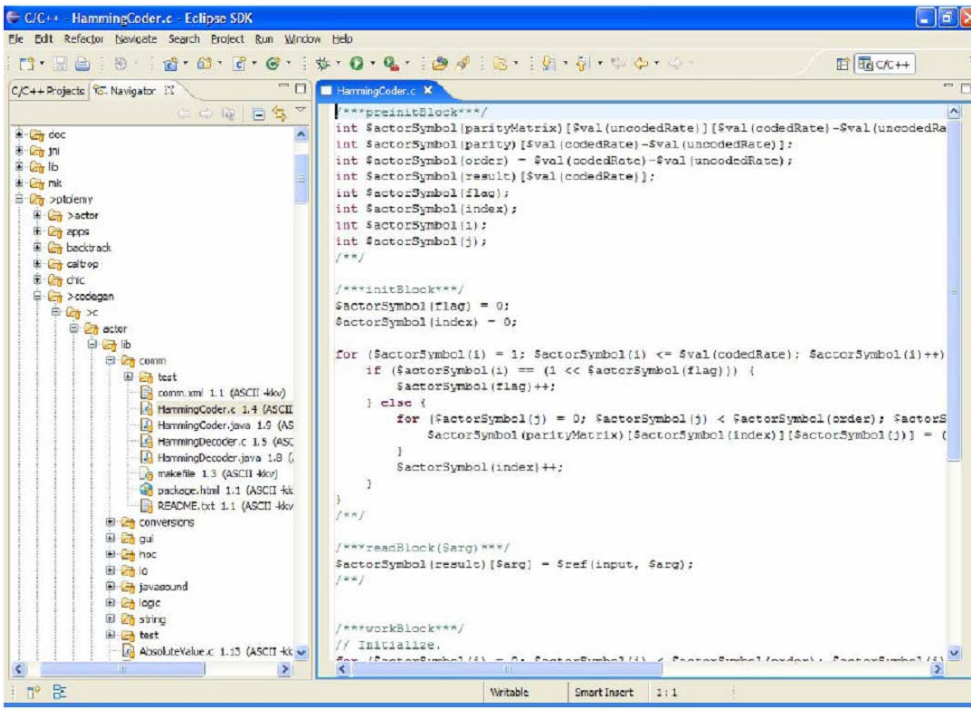

Figure 9.0 Eclipse CDT with Ptolemy II "codegen" "HammingCoder" Example Code

Without the Ptolemy II codegen automation tools, the C/C++ code must be generated by hand and the Eclipse tool suite, along with other open-source compiler tools, is a reasonable approach for design entry directly as C/C++ code. To also use Eclipse for VHDL/Verilog development, "plugin" is available at: http://sourceforge.net/projects/veditor/ or for a licensed tool at: http://simplifide.com/html/general/download.htm

The Eclipse tool is a very reasonable, general purpose design entry that has a capability to link its tool chain" to the open-source compiler resources, as well as proprietary tools as well. It is available in versions for multiple operating systems including the Microsoft OS suite, as well as Linux. In the Linux environment, the tool chain installation of the open-source tools is accomplished by a few selections to the tools supplied with the Linux distribution. To enable the Eclipse tool to compile in a Windows environment, a tool set like MinGW and MinDB needs to be obtained and installed prior to its selection from Eclipse. The MinGW

tool suite is available from: http://wascana.sourceforge.net/ Many of the commercially available tools are built on the Eclipse software as a base.

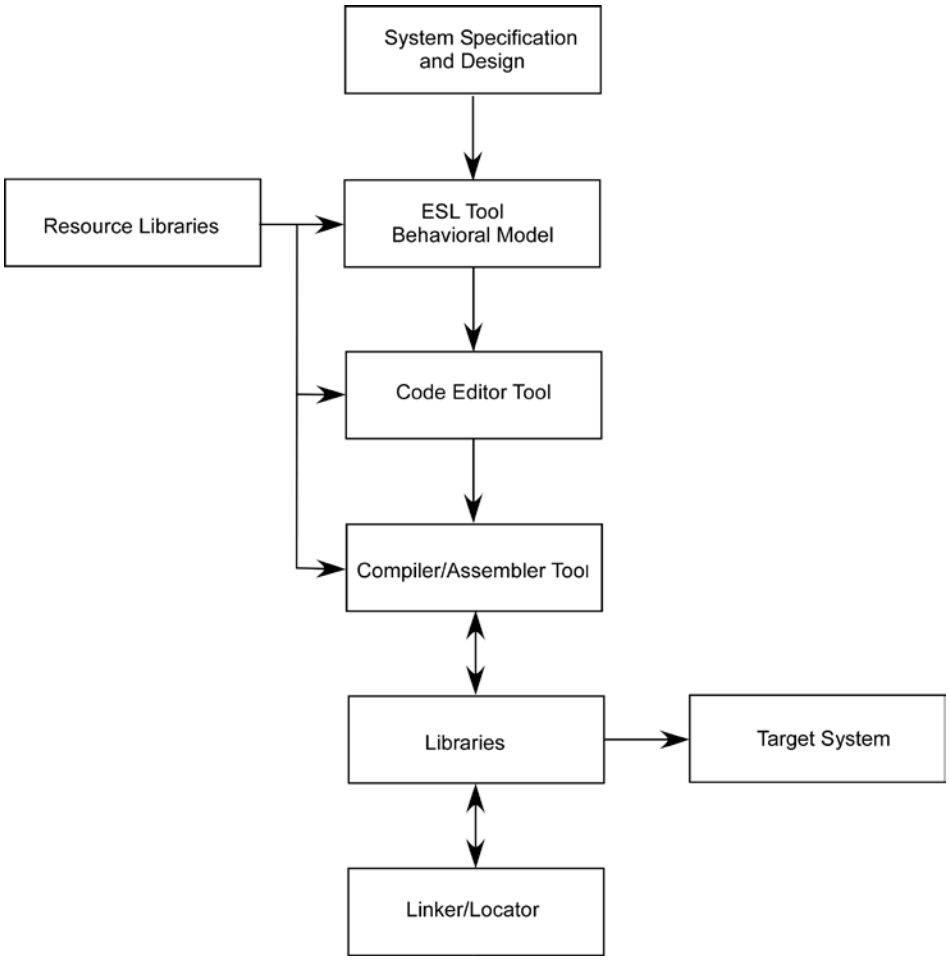

Figure 9.1 A Design Flow using Typical Tools

Computer Language Tools: The Eclipse tool suite supports other languages as well as the C/C++ and VHDL/Verilog examples discussed above. Fortran and Pascal have some levels of support and compilers for target hardware including the ARM family of processors, and are required to be specified in the tool chain for correct object code to be produced using the Eclipse "Run" specification for object code generation.

The general organization of code development for any digital computer system follows a similar design flow, with the possibility of original language input from many different source languages. C/C++ are popular today, but there is a great deal of Fortran code available

from signal processing applications that may need to be included, as well as the VHDL/Verilog for FPGA/ASIC specification and synthesis.

We have discussed the Top-Down design elements of the Engineering Systems Level (ESL) using Ptolemy II and Simulink™ tools and now add some detail surrounding the information flow through tools that may be hidden by automation. The resource libraries may be expanded by interactions, but are generally a resource of design elements, code examples, Real-Time OS components, Application Programming Interface (API) elements, compiler resources for particular target hardware, and prior higher-level sub-system designs. The use of a library of resources at the higher-level is one of the main enablers of software re-use.

We have seen that automation tools such as the Ptolemy codegen and the MathWorks RT-Workshop tools can provide code in C/C++ and/or VHDL/Verilog language formats for introduction into a code editor tool such as Eclipse or other similar tool. The editor provides a standard, high-level interface to the compiler/assembler so that tool can provide code translation to the less-readable, relocatable, target-specific, assembly language form.

Libraries are built of relocatable code and more specific linked executable object code and stored in the target-specific library for placement in the target system memory. The linker/loader is used to convert the relocatable code into the executable object form and put the result back into the library. The linker/loader is useful for configuring systems with differing capabilities from an assortment by selection. System changes between different input/output devices are easily accommodated, for example.

Operating System Libraries: There are several operating systems available from QNX, Wind River, Green Hills, and others, as well as open-source Linux variants. In the open-source arena, there is also the possibility of using “hard” real-time OS capabilities for those embedded systems with the need for minimum response times.

One useful configuration is the combination of Linux with the RTLinuxFree version from Wind River. RTLinuxFree version is available from: http://www.rtlinuxfree.com/ The RTLinuxFree is a small real-time kernel program that runs in the Linux environment and provides the interrupt-driven behavior of the real-time system and leaves most of the underlying Linux capability intact. In this fashion, slower functions like Geographical Information System, database, and some display interface computation can be assigned to Linux capabilities and fast-response, interrupt driven behaviors serviced by the RTLinux kernel. Linux runs as the lowest-priority, idle-task inside RTLinux.

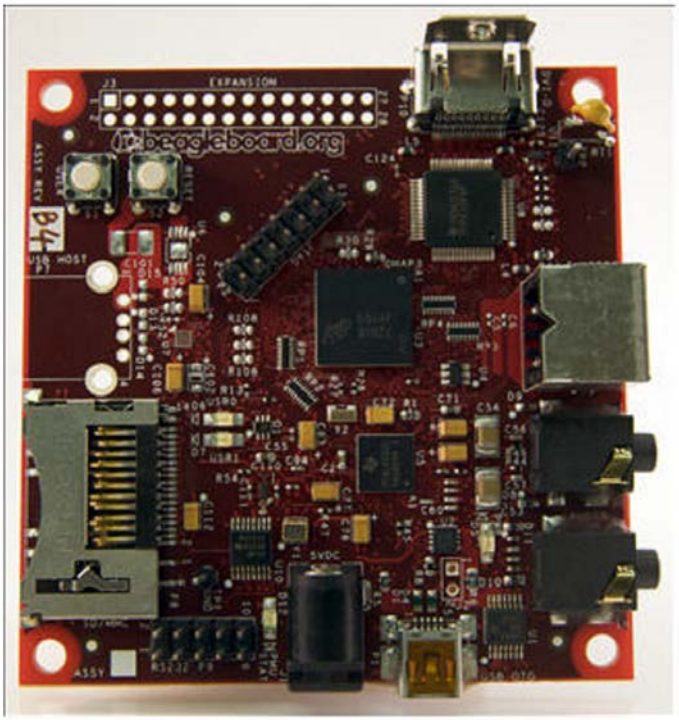

Figure 9.2 BeagleBoard with the TI OMAP Processor; ARM and DSP

Open-Source Design-Entry Systems: We have discussed the Ptolemy II ESL tool, its codegen capability, and the Eclipse editor/developer system in the context of design entry for an open-source tool chain solution. We have also discussed the Linux operating system and the inclusion of the RTLinuxFree for "hard" real-time support. Once the code is generated for a general-purpose platform such as the BeagleBoard that uses the TI OMAP processor, however, a tool chain link must be made to the compiler resources that produce the executable object code for the hardware resources. The BeagleBoard presents some additional issues because the hardware includes the DSP core that needs its own development support. Information on the TI OMAP 35x Linux PSP can be obtained at:
http://focus.ti.com/lit/an/sprs640/sprs640.pdf

In addition, information about code development for the DSP, using the TI Code Composer Studio version 3 (CCS3) using the DSP/BIOS RTOS can be obtained at:
http://e2e.ti.com/support/dsp/omap_applications_processors/f/42/t/34040.aspx

Unfortunately, as of this writing, the DSP tools are not "Linux-aware" and some effort is required to "meld" the code from two different development flows to make the application-space overlap.

Proprietary Design-Entry Systems: In parallel with the open-source tools, the MathWorks and others offer suites of commercial tools to support embedded design activities. In section 6.0 above, we have presented the Simulink™ tool from MathWorks that enables the ESL block diagram simulation of system behavior. To support the migration of that behavior to an embedded design implementation, MathWorks offers the "Real-Time Workshop™ (RTW)" tool for generation of ANSI-compliant C-code from MatLab™, StateFlow™, as well as Simulink™ models to be targeted to an embedded processor. MathWorks also offers the Simulink™ HDL Coder that generates VHDL/Verilog HDL code models that can be targeted to standard FPGA components or ASIC designs.

To support designs with components written in the "C" programming language, MathWorks offers the PolySpace™ tool that performs a "semantic" analysis check on the code. It locates and indicates code that can generate run-time errors including division-by-zero, square-root of negative arguments, attempted data-structure access outside declared bounds, and other difficult-to-find coding problems. The PolySpace™ should be considered a "best-practice" tool for preparing embedded "C" code.

One IC component company, Cypress Semiconductor, has chosen to provide an entirely different design-entry methodology for the "Programmable System on Chip (PSoC)" product line. Cypress Semiconductor has made arrangements with Keil, a division of ARM, to provide a proprietary set of compilation tools to complement it product line in the PSoC Designer™, PSoC Creator™, and PSoC Programmer™ packages that they supply.

The interface to the PSoC Creator™ design entry tool is shown in figure 9.3 below. There are numerous tutorials available that teach the use of the free, but licensed/registered tools. The design entry tool combines a schematic/block diagram entry with the code development for the microprocessor component of each member of the PSoC family. The ARM Cortex-M3 is the target processor included in the PSoC® 5 we use in a later example here, but other processors are supported in the common tool suite for the other members of the PSoC family shown in figure 9.5 below. The PSoC family is distinctive because it contains a fully featured microprocessor along with configurable digital and analog components. The PSoC Creator™ software supports a drag-and-drop style construction of system block diagrams that are converted into the PSoC configuration instructions by the tools. An extensive library of

component sub-systems is provided as part of the support. The design flow does not include a simulator, but rather depends on the actual hardware for emulation during design debug activities.

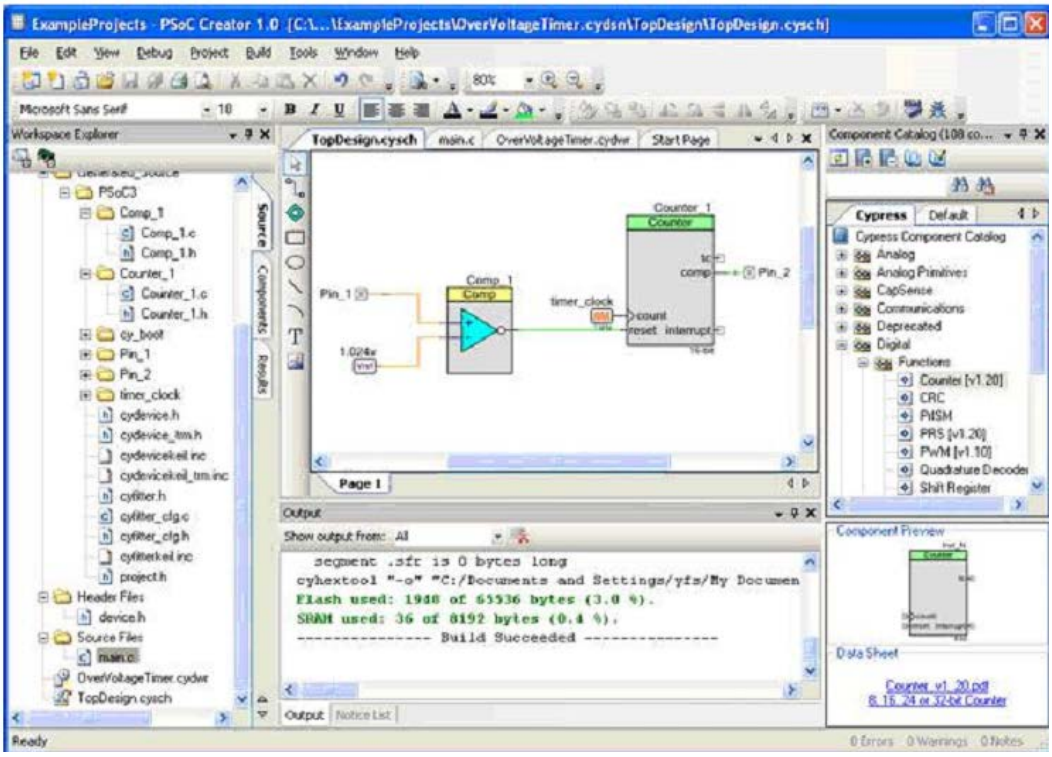

Figure 9.2 Cypress PSoC Creator™ Software Interface

In addition to the PSoC Creator™ software, the PSOC Programmer software, shown in figure 9.3 below, is also required to load the "FLASH" memory code space with the executable and configuration code to complete the design.

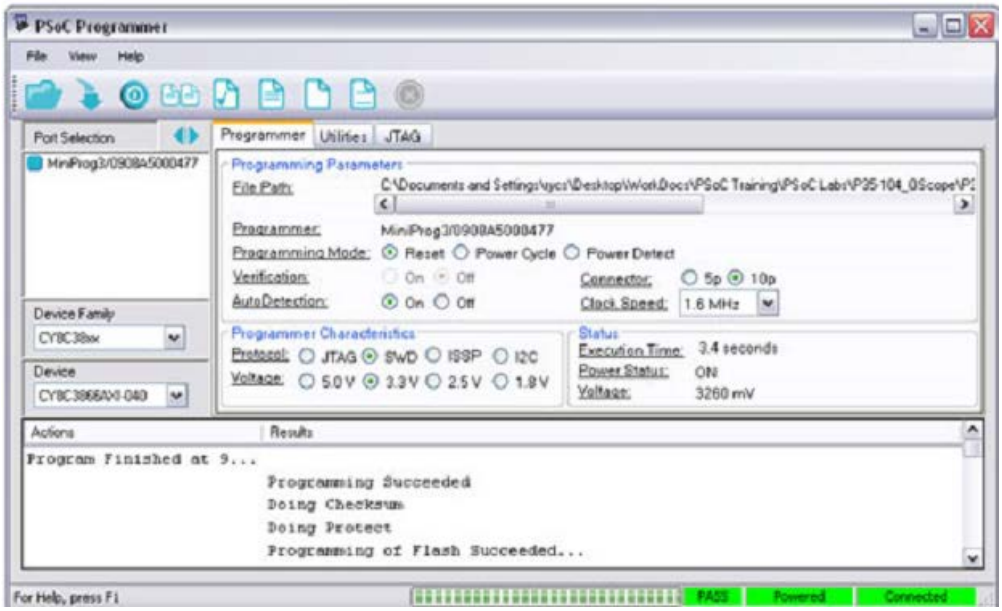

Figure 9.3 Cypress PSoC™ Programmer Software Interface

The on-board microcomputer code is developed in C/C++ in a parallel activity using the Keil µVision4™ IDE and support tools as shown in the design flow in figure 9.4 below.

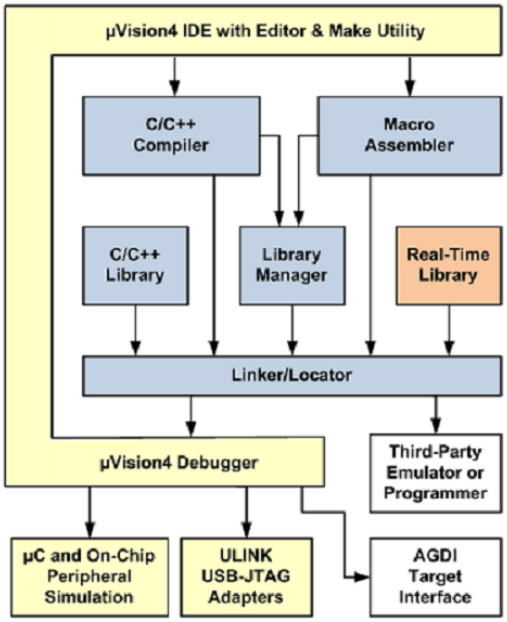

Figure 9.4 Keil™ Software Development Process Flow

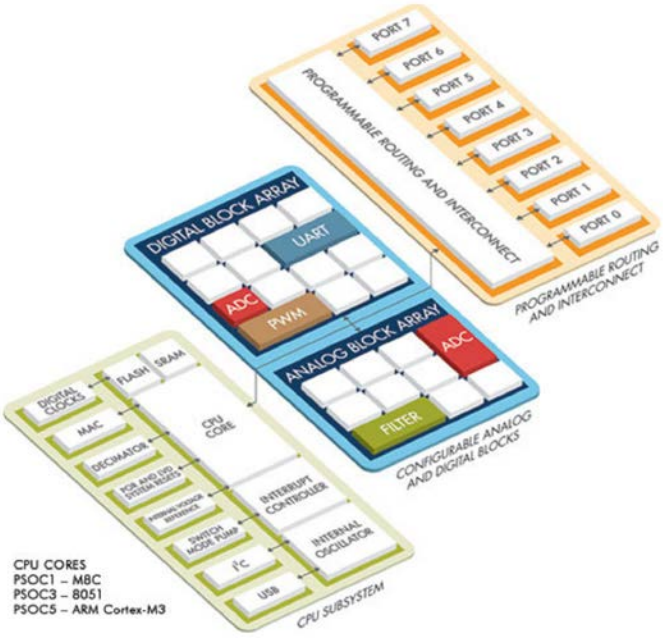

Figure 9.5 Cypress PSoC™ Family Architecture

As shown in figure 9.5 above, the PSoC family of products contains a microcomputer, a set of configurable digital blocks, a set of configurable analog blocks, as well as configurable Input/Output functions from within the context of a single integrated circuit.

Figure 9.6 Cypress PSoC™ First Touch Bundle

As shown in figure 9.6 above, the PSoC First Touch is a "bundle" with the experimenter board, USB cable, a complete set of development software, and even a battery to support hand-held operation included.

10.0 Example Embedded System

We have chosen a single design example, but show different design-flow derivations for both embedded system example hardware platforms. We begin with the PSoC5 example and explore the design process and follow with a contrasting BeagleBoard implementation.

PSoC5 Example: The PSoC5™ "First Touch" board shown in figure 10.0 below, has the components on-board to support a demonstration of an interface with an accelerometer. It includes the PSoC 5 device itself including the on-chip resources shown in figure 9.5 above. An accelerometer and sufficient other resources to support power from either a USB port or by attachment of a 9V battery to the board are also shown and labeled below.

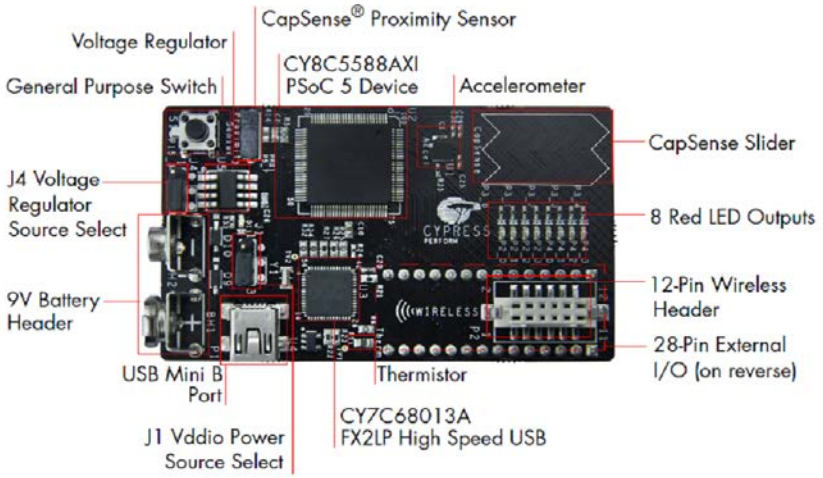

Figure 10.0 Cypress PSoC5™ First Touch System Experimenter Board Detail

Design-flow environments for the PSoC5 include the "First Touch" system experimenter board shown in figure 10.0 above, as well as a larger PSoC development kit system (CY8CKIT-001). Both support the Cypress PSoC 5 Integrated Circuit itself. The PSoC development kit system also supports plug-in daughter boards for each of the other PSoC components and has additional software specific to each of the other products all contained in a common environment. For the PSoC3 and PSoC5 families, the tools are essentially identical, except for the microprocessor object code differences. The PSoC3 employs an 8-bit 8051 microprocessor and the PSoC5 employs a 32-bit ARM microprocessor. The PSoC1 employs a proprietary microprocessor and requires a somewhat different set of software.

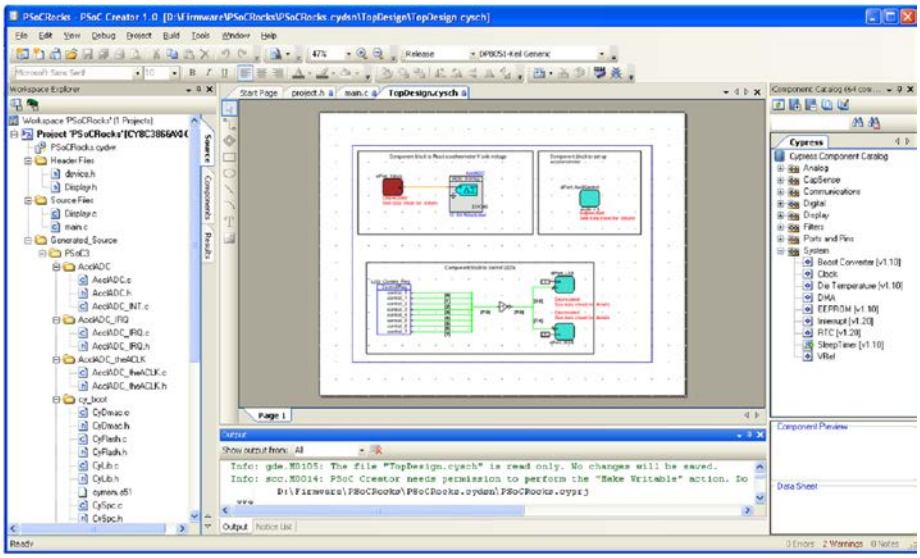

Figure 10.1 Cypress PSoC5™ Creator Accelerometer Demonstration

The Cypress PSoC 5 Integrated Circuit and each of the other PSoC components, are supported by the common design-flow that includes a “First Touch” design example and a software tool suite with the PSoC Creator graphical schematic/block diagram design entry tool as shown in figure 10.1 above for the PSoC3 and PSoC5, with a common C-code programming window as shown in figure 10.2 below. The PSoC Designer software is required for the PSoC1 parts.

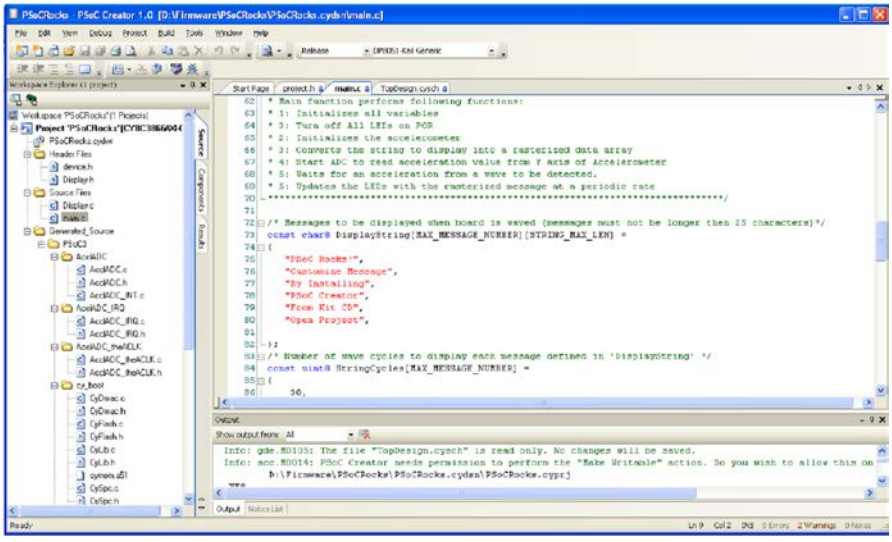

Figure 10.2 Cypress PSoC5™ Default C Code for ARM processor

A library of hardware/software components is supplied for the capabilities internal to each IC in each family, as well as a number of API components for peripheral components. A partial list of components for selection can be seen in the right-hand window of the PSoC Creator GUI shown in figure 10.1 above. In both figures 10.1 and 10.2 above, the left-hand window of the PSoC Creator shows the entire project hierarchy.

The following is the C-code program for the accelerometer and display program:

```
void main()
{
  uint8 RasterCount;    /* The current raster column data to the LEDs */
  uint8 MessageNumber;  /* The current message number to display */
    uint8 Cycles;       /* The number of hand waves a message was shown */
    uint8 RasterLength; /* The length of the message's raster array */

    /* Intitalize hardware */
    CYGlobalIntEnable;        /* Enable global interrupt */
    LED_Control_Reg_Write(0); /* Turn off the LEDs on PORT2(pin 0-3) and
PORT4 pin(0-3) */
    dPins_AcclControl_Write(ACCL_SET_CONTROL);  /* Set control bits of
Accelerometer */
    AcclADC_Start();

    /* Continuously rotate through display messages when waved */
    while(1)
    {
        for(MessageNumber = 0; MessageNumber < MAX_MESSAGE_NUMBER;
MessageNumber++)
        {
     /* Convert current message string into rasterized data array */
            RasterLength =
LED_StringProcess(DisplayString[MessageNumber]);

     /* Display message the required number of hand wave cycles */
            for(Cycles = 0; Cycles < StringCycles[MessageNumber];
Cycles++)
            {
     /* Wait for ADC conversion to complete and for + acceleration to
trigger start */
                do
                {
                    AcclADC_StartConvert();
                    AcclADC_IsEndConversion(AcclADC_WAIT_FOR_RESULT);
                }while(AcclADC_GetResult16() > ACCEL_TRIGGER);

     /* Generate initial delay between acceleration trigger and start of
display output */
```



```
     /* May be adjusted for wave speed in conjunction with RASTER_DELAY
value */
                LED_Delay(INITIAL_DELAY);

     /* Output each raster column sequentially to the LEDs with */
     /* Delay between each raster column. Sets optimum hand wave speed */
                for(RasterCount = 0; RasterCount < RasterLength;
RasterCount++)
                {
                    LED_Delay(RASTER_DELAY);

LED_Control_Reg_Write(DisplayRasterTable[RasterCount]);
                }

    /* Wait for ADC conversion to complete and for - acceleration to
signal wave end */
                do
                {
                    AcclADC_StartConvert();
                    AcclADC_IsEndConversion(AcclADC_WAIT_FOR_RESULT);
                }while(AcclADC_GetResult16() < (WAVE_END_VALUE -
ACCEL_TRIGGER));
            }
        }
    }
}
```

Listing 10.0 Cypress PSoC5™ Default Accelerometer C-Code© for ARM processor

The code shown above in listing 10.0 is combined with the declarations for the message string shown in figure 10.2 above and other declarations at that location and compiled using the free Keil™ tools supplied.

Compilation is achieved using the pull-down menu under “Build” with the project name as the selection item as shown in figure 10.3 below.

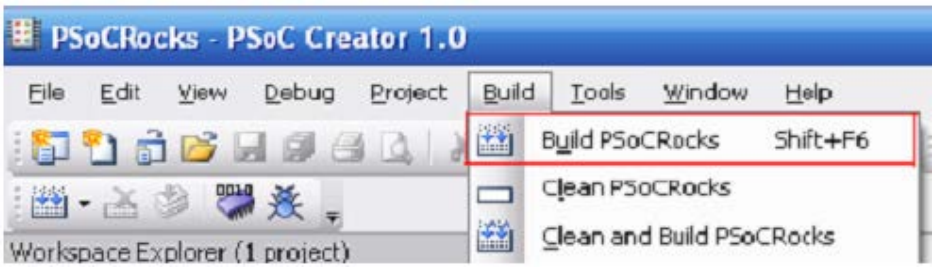

Figure 10.3 Cypress PSoC5™ C Code Compile for ARM processor

The PSoC Creator tool uses the "build" command from the pull-down menu as shown in figure 10.3 above to compile the C-code from the source listing to object code. It obtains the C-code referenced in the program calls from the library, compiles and links code from the library, and employs the built-in tools to write the resulting executable object to the "Flash" memory on the PSoC5, in this case on the First Touch board. In all environments, the tools perform the same functions, loading the PSoC on board Flash memory in the process.

After the tools perform the sequence of tasks outlined above, a message about the success or failure is displayed in the tool's output window shown in figure 10.4 below

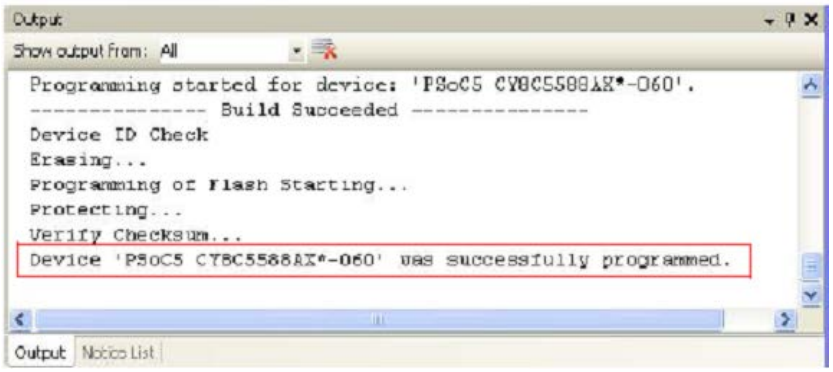

Figure 10.4 Cypress PSoC5™ C Code Compile and Load Success Message

The "First Touch" PSoC5 board provides the demonstration platform for the PSoC5 and supports tutorial editing of the accelerometer-driven LED display with re-compilation to show a user-replaceable message. Notice that in the design-entry window of the PSOC Creator software shown in figure 10.1 above, a schematic symbol is shown for a **ΔΣ** converter block. The signals are routed to the resources and the C-code is compiled and loaded into the FLASH memory for the ARM microprocessor to execute as part of the compilation process.

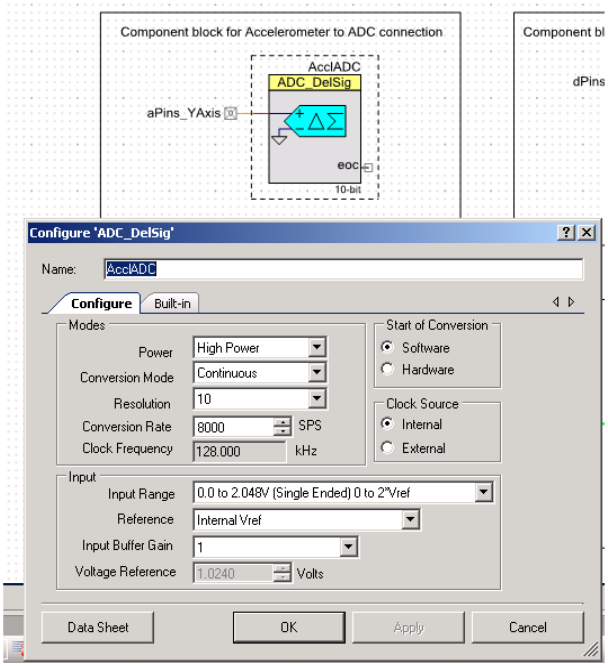

Figure 10.5 Cypress PSoC5™ "Native" ΔΣ Converter and Configure Dialog

The **ΔΣ** converter is a "native" function within the PSoC5, but can be realized in several different forms including that block shown or other realizations composed of different combinations of other PSoC5 internal digital and analog blocks. Different architecture alternatives can be explored to meet different goals by re-programming the part. The "native" **ΔΣ** (**ADC_DelSig**) converter function within the PSoC5 can be configured to realize many implementation choices as shown in figure 10.5 above. The resolution can be selected up to 20-bit accuracy at slow clock speeds and lower accuracies at higher clock speeds.

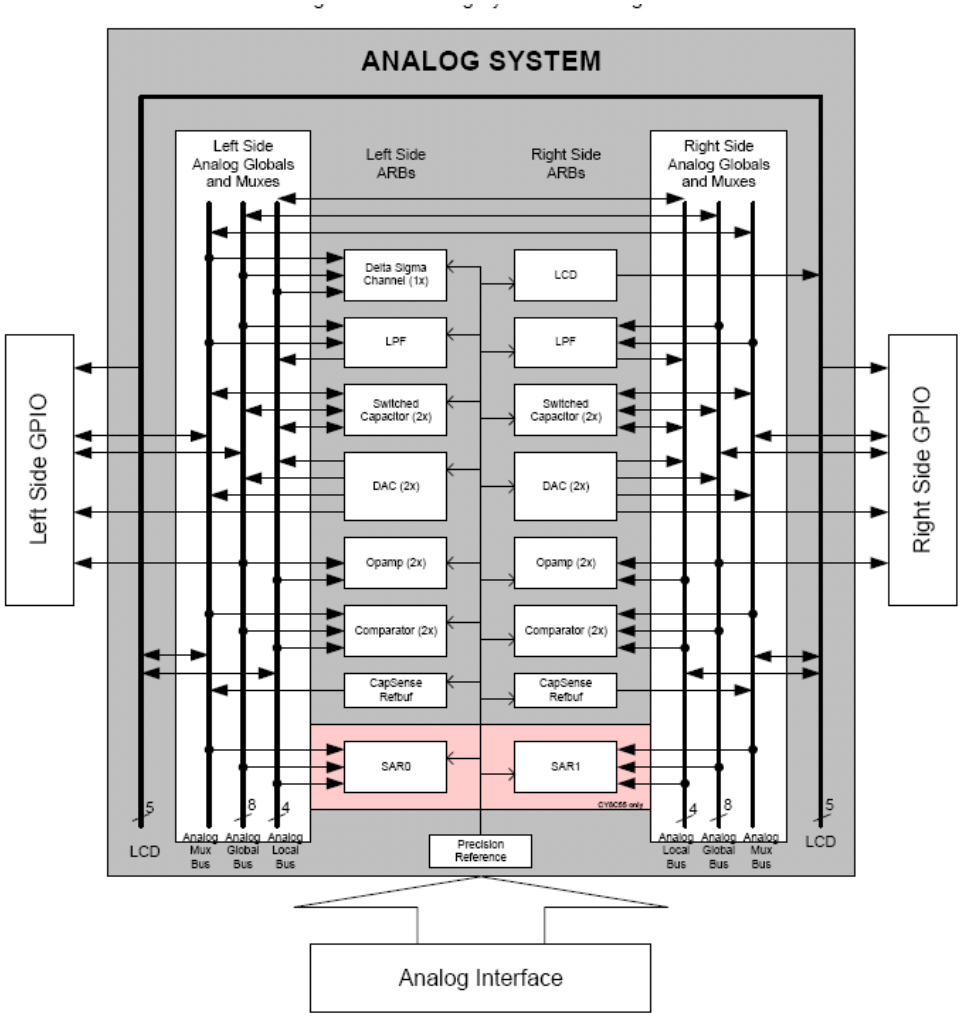

Figure 10.6 Cypress PSoC5™ Configurable Analog Blocks

Within the "Configurable Analog Blocks" of the PSoC5 ™ with the detail shown in figure 10.6 above, we see the single configurable **ΔΣ** Analog-to-Digital converter in the upper left block. This is the block we showed in figure 10.5 above with the configuration dialog box shown. The "Configurable Analog Blocks" of the PSoC5 ™ include an LCD driver block, two Low Pass filters, four configurable switched-capacitor stages, four Digital-to-Analog (DAC) blocks, four configurable OpAmps, for Comparators, a pair of Capacitance-sensing circuits, a pair of Successive Approximation Analog-to-Digital converters, and a precision voltage reference circuit.

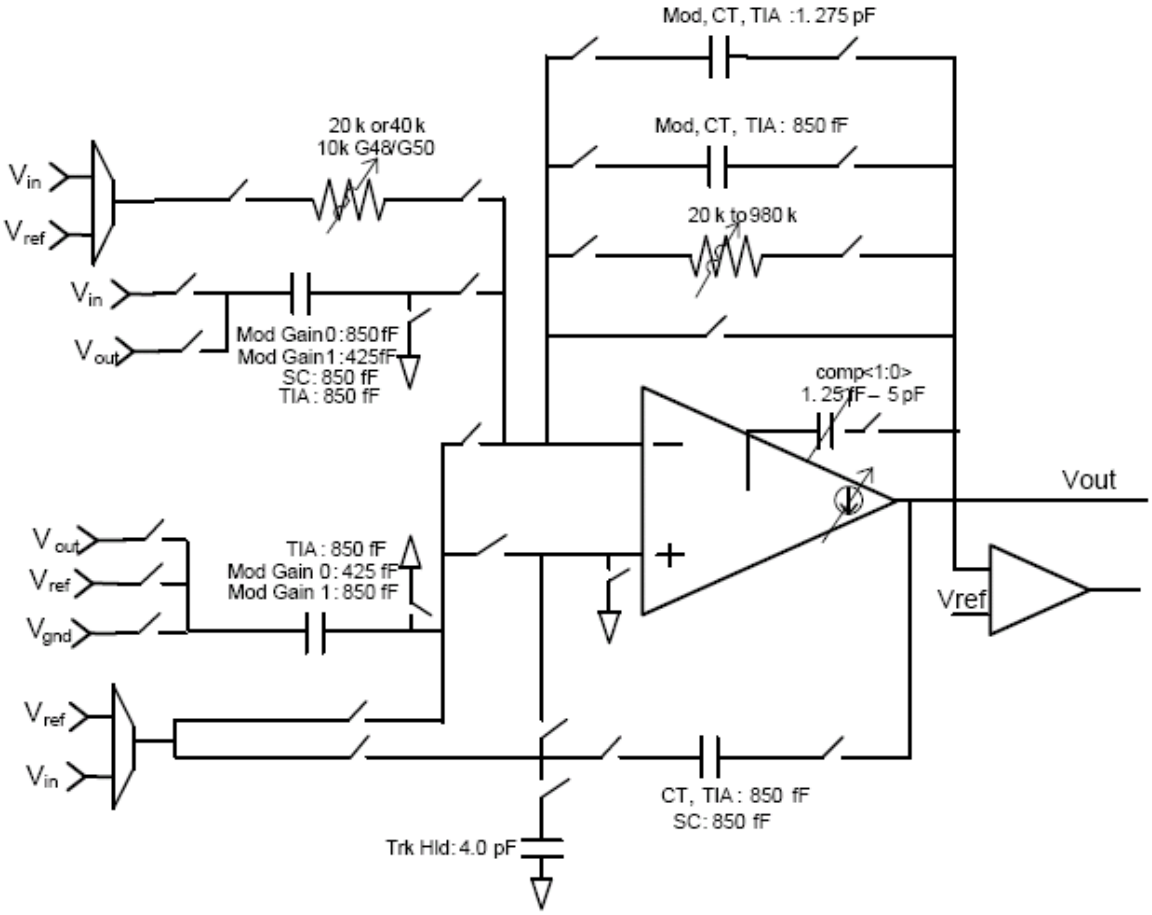

Figure 10.7 Cypress PSoC5™ Switched-Capacitor Filter Block Detail

Each Configurable Analog Block shown in figure 10.6 above has programmability similar to the **ΔΣ** Analog-to-Digital converter block and a detail for the switched-capacitor block is shown in figure 10.7 above with a detail of some of the possible configuration choices for each of the switched-capacitor blocks.

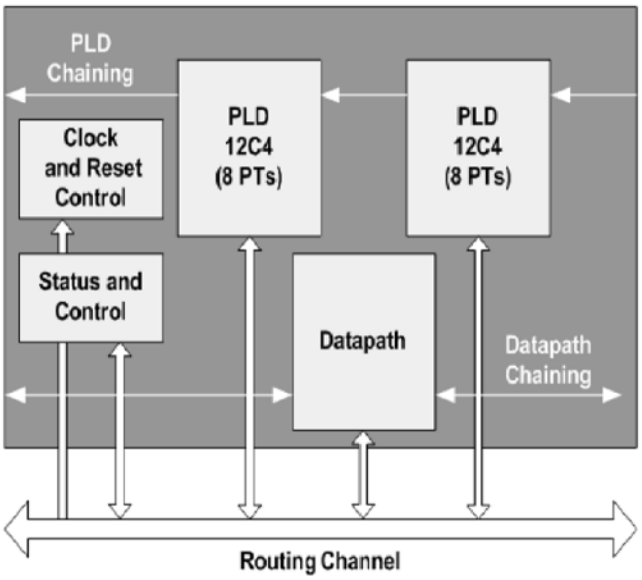

Figure 10.8 Cypress PSoC5™ Universal Digital Block Components

Within the "Configurable Digital Block" first shown in figure 9.5 above, we have 24 instances of the "Universal Digital Blocks (UDB)." Within each UDB of the PSoC5 ™ we see the Datapath, the Clock and Reset Control functions, the Status and Control functions, and "Programmable Logic Device (PLD)" blocks, as shown in figure 10.8 above. The PSoC5 on the "First Touch" board has 24 UDB block instances available.

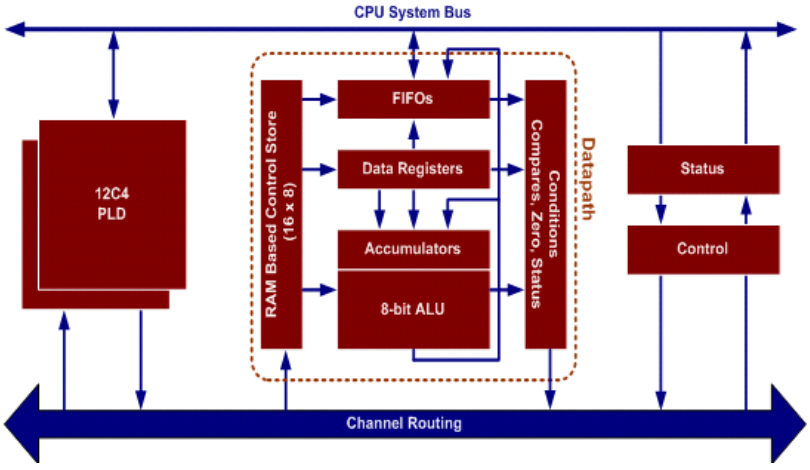

Figure 10.9 Cypress PSoC5™ UDB NanoProcessor Sub-System Component

Within the Datapath of each of the 24 UDB blocks of the PSoC5 ™ we see in figure 10.9 above a “NanoProcessor” that can be programmed from a Verilog-coded program to perform many different possible functions.

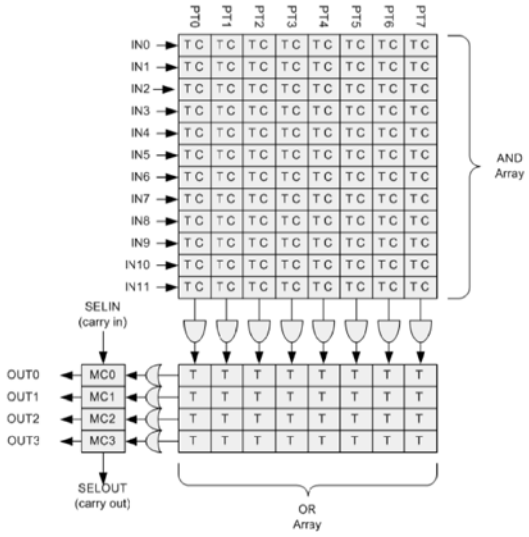

Figure 10.10 Cypress PSoC5™ Programmable PLD Sub-System Component

Within each UDB of the PSoC5 ™ we see in figure 10.10 above an AND-OR programmable Boolean “Programmable Logic Device (PLD)” that can be programmed to perform many different combinational logic functions.

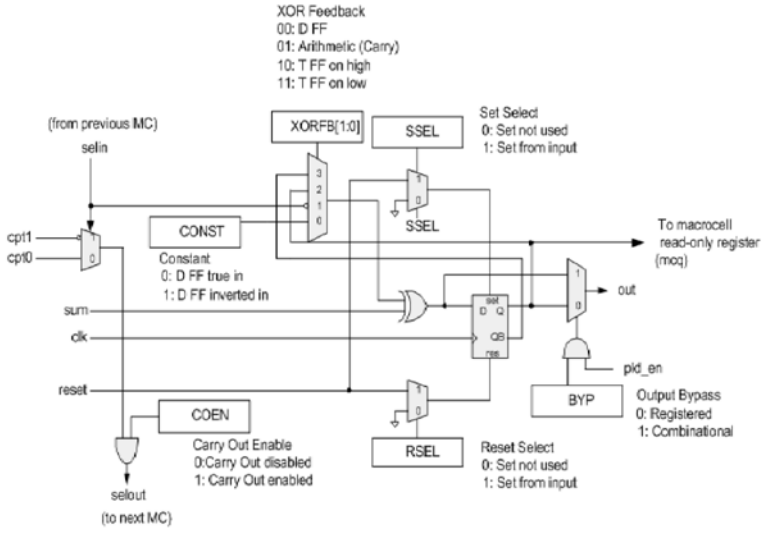

Figure 10.11 Cypress PSoC5™ Programmable Macro-Cell Sub-System

In figure 10.11 above, we see an example of a "Programmable Macro-Cell" that is utilized in many of the digital components accessed from the pull-down library list and with a drag-and-drop, placed in a system diagram. The Macro-Cell uses some, but not all, of the resources of a UDB and is configurable within specific component's dialog boxes, or through a set of Verilog code calls.

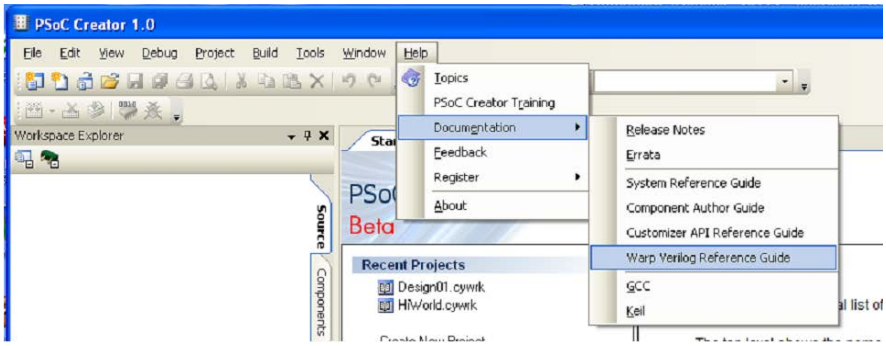

Figure 10.12 Cypress PSoC5™ Verilog Tools Support Documentation

In figure 10.12 above, we see the pull-down "Help" menu path to access the documentation for the Verilog tool that is built-in to the PSoC Creator software. Additional tutorial/training resources are available directly from Cypress after a brief sign-up procedure at: http://www.cypress.com/?rID=40547

BeagleBoard Example: The BeagleBoard that was shown in figure 9.2 above, is supported by beagleboard.org for both hardware and software items. You should download the system reference manual from: http://beagleboard.org/static/BBSRM_latest.pdf

Besides the BeagleBoard itself, there are a number of other components/devices/cables you will need. They are listed and discussed at: http://code.google.com/p/beagleboard/wiki/BeagleBoardShoppingList

You will need the capability of programming SD cards and attaching them to the BeagleBoard. You have numerous alternatives for a user interface, including an LCD panel, a video interface, and a powered-USB interface that allows support from a "host" PC system.

A Keil software development system that is based on the open-source Eclipse software can be downloaded as a free trial (until 30 Sept 2010) from: www.keil.com/ds5 . The Keil software is very similar to that provided free with the PSoC5 above.

You should download a Linux distribution such as the Ubuntu version available at: http://elinux.org/BeagleBoardUbuntu

Follow the directions at that site for installing the software on the SD card.

The DSP capabilities can be added and supported by downloads from: http://www.elinux.org/BeagleBoard_Ubuntu_%26_DSP_From_Sources

The DSP is not required for building the example, but DSP capability is one of the main features and a primary reason for exploring BeagleBoard hardware.

We recommend that you download and deploy Open RTLinux from the WindRiver site at: http://www.rtlinuxfree.com/

The RTLinux will enable you to install and run all device API software supported by the Ubuntu Linux, but add the interrupt handling capabilities of a hard real-time system.

To enable the control of an accelerometer, you will also need to obtain or construct an experimenter board to support/interface to the accelerometer chosen. A three-axis accelerometer (Kionix, KXSC7-2050) is supplied with the PSoC5 board in the example above, so that particular manufacturer/model could be chosen for a direct comparison. A number of boards with support such as the one shown in figure 10.5 below, are available directly from Kionix at the site: http://www.kionix.com/sensors/evaluation-kits.html

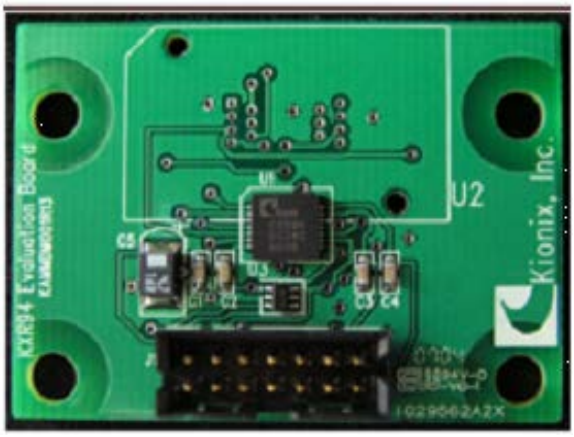

Figure 10.13 Kionix Accelerometer Demo Board

Armed with the hardware and software recommended above, the ΔΣ controller software can be constructed as C-Code from within the Eclipse or Keil environment and compiled for the target system.

C-code similar to that shown for the PSoC5 example in listing 10.0 can be used as a template for development with the addition of driver routines appropriate for the peripherals employed/attached to the BeagleBoard hardware.

At this point in the discussion, it is worthwhile to discuss why a designer would consider the use of the BeagleBoard with all the levels of complexity versus using the PSoC5 with its "easy-to-use" environment. There are two primary reasons for considering the BeagleBoard in preference to the PSoC5: first, the DSP co-processor on the OMAP/BeagleBoard offers very high speed computation for applications including video and ultrasound processing, and second, the latest release of the BeagleBoard takes the clock speed to the Giga-Hertz realm. In short, the BeagleBoard offers a large speed advantage over the PSoC5 solution. However, the PSoC5 offers the ease-of-use for quick prototypes and an introduction to the novice developer.

Following the example to its conclusion, we recommend that a MathWorks took suite including RT-Workshop and Simulink-HDL coder be obtained to make the C-code development tractable from a Simulink ESL model.

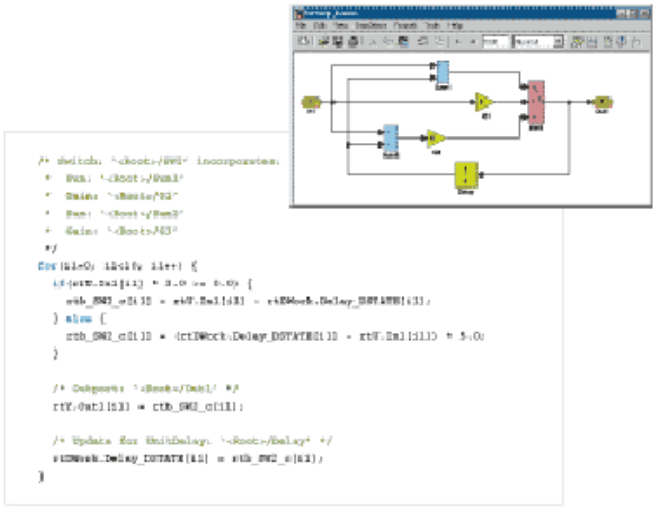

Figure 10.14 MathWorks RT-Workshop™ Tool for Simulink™ to C-Code Generation

The MathWorks site for RT-Workshop is at: http://www.mathworks.com/products/rtw/

Figure 10.15 MathWorks Simulink-HDL™ Tool for Simulink™ to HDL Generation

As an adjunct to RT-Workshop, the Simulink-HDL coder with several screens shown in figure 10.15 above is available to generate Verilog and VHDL code from Simulink models. This tool eases the effort of generating synthesizable code for target ASIC and FPGA component designs to augment the BeagleBoard hardware (much as you added the accelerometer). Additional tools are available from MathWorks and others to compile the resulting code into a form usable for FPGA programming.

MathWorks Simulink-HDL coder is at: http://www.mathworks.com/products/slhdlcoder/

C-Code Checking: The BeagleBoard shown in figure 9.2 above is supported by different possible programming tools in both open-source and commercial offerings with substantial support. Likewise, the PSoC5 and its entire family are supported by a similar design-flow set of code development tools. Regardless of the source of any C-code for the target system, whether obtained from available source code files or written for the purpose, there is substantial verification required to ensure proper operation of the resulting object code prior to it being compiled and loaded into the target system hardware.

One “best-practice” tool that performs a “semantic” analysis on ANSI-Standard C-code is the PolySpace™ tool offered by MathWorks. PolySpace™ performs checks for run-time errors including variable initialization, division by zero, underflow/overflow conditions and other such error generating conditions.


```
70        static void Pointer_Arithmetic ()
71        {
72          int tab[100];
73          int i, *p = tab;
74
75          for(i = 0; i < 100; i++, p++)
76            *p = 0;
77
78          if(get_bus_status() > 0)
79          {
80             if(get_oil_pressure() > 0)
81               *p = 5; /* Out of bounds */
82             else
83               i++;
84          }
85
86          i = random_int();
87          if (random_int())  *(p-i) = 10;
88
89          if (0<i && i<=100)
90          { p = p - i;
91            *p = 5;      /* Safe pointer access */
92          }
93        }
```

Listing 10.1 MathWorks PolySpace™ C-code Syntax Check

In listing 10.1 above, the PolySpace™ tool indicates in green as in lines 75, 76, 78 and others, any code that is "safe" and cannot cause a run-time error. In line 81, a "bug" is indicted in red, along with the comment that the instance obtains a pointer that is "out of bounds" and must be corrected or the program will fail at run-time. In line 83, an instance of unreachable/dead code is indicted in gray. . In line 87, a warning is indicated in orange.

With the liability concerns of the Professional Engineer faces in project design responsibility, a best-practice code check tool is recommended to help ensure correctness. The tool does not absolve the engineer of responsibility, but assists in complete checking.

11.0 Summary and Conclusions

We have traced the background history leading to current "Embedded Systems" from the introduction of system classification concepts through the development of digital computer

concepts, and first-practice digital computer embedded systems leading up to the current methodologies, tools, and practice involved in the design of Embedded Systems.

The digital computer hardware was traced from the vacuum-tube era through the advances of solid-state technology to today's integrated circuits. The software was shown in parallel with introduction of abstract languages, operating systems and "hard" real-time software, program code and library practices, through C-Code editing, compilation, and system building. Hardware Design Language (HDL) concepts were introduced in the context of adding digital functionality to the included microcomputer capabilities.

The Top-Down design methodology was introduced in the context of Embedded Systems development using both open-source and proprietary tools from an Engineering System Level (ESL) through an implementation at the demonstration-board level with two different Embedded System Integrated Circuit (IC) examples. The design-flow through the use of tools was discussed prior to the example use of those tools so that the "big picture" was seen first. A simple Delta- Sigma ($\Delta\Sigma$) analog-to-digital converter was used as an analog/digital or mixed-signal example with some modeling and simulation relating to the analog functions and some relating to the digital functions. The use of the top-down approach made the system simulation of behavior clear prior to the "binding" of the functions to analog or digital components.

Two different implementation hardware boards: the BeagleBoard featuring a Texas Instruments (TI) OMAP™ IC with an ARM processor and TI proprietary Digital Signal Processor (DSP) onboard, and Cypress Semiconductor PSoC5™ with an ARM processor and proprietary configurable analog and digital blocks on a "First Touch" board were introduced as a hardware target candidates.

Software development tools were introduced and representative usage shown for both open-source and proprietary tools using a C/C++ path in both an open-source Linux environment and a MicroSoft Windows™ environment. Code generation automation was discussed and manual code entry discussed in the context of the tools.

The example system was explored in detail from within the context of the Cypress Semiconductor PSoC5™ "First Touch" environment with some alternatives introduced. The steps necessary and the code example for the PSoC5™ "First Touch" environment was referenced, as well as the necessary hardware and software requirements to enable the example in the BeagleBoard to be shown and links provided to obtain the accelerometer,

display, cables, and software required. The MathWorks PolySpace™ semantic run-time code checker software was shown with illustrations of the checking capabilities.

12.0 Copyright Attachments

Copyrighted material from Cypress Semiconductor used by permission.

Ptolemy II Version 7.0.1
Below is the copyright agreement for the Ptolemy II system.