

RAPID SOC PROOF-OF-CONCEPT FOR ZERO COST

JEFF MILLER, PRODUCT MARKETING AND STRATEGY, MENTOR GRAPHICS
PHIL BURR, SENIOR PRODUCT MANAGER, ARM



A M S D E S I G N & V E R I F I C A T I O N

W H I T E P A P E R

www.mentor.com

THE SCENARIO

Your company provides analog/mixed-signal (AMS) and sensor-based ICs, but your best customer wants you to create a system on a chip (SoC) that includes a digital processor (Figure 1). With little experience with digital processors, you need to quickly provide a proof-of-concept to your customer that shows the viability of this new IC in the next few days. And, you have very little budget.

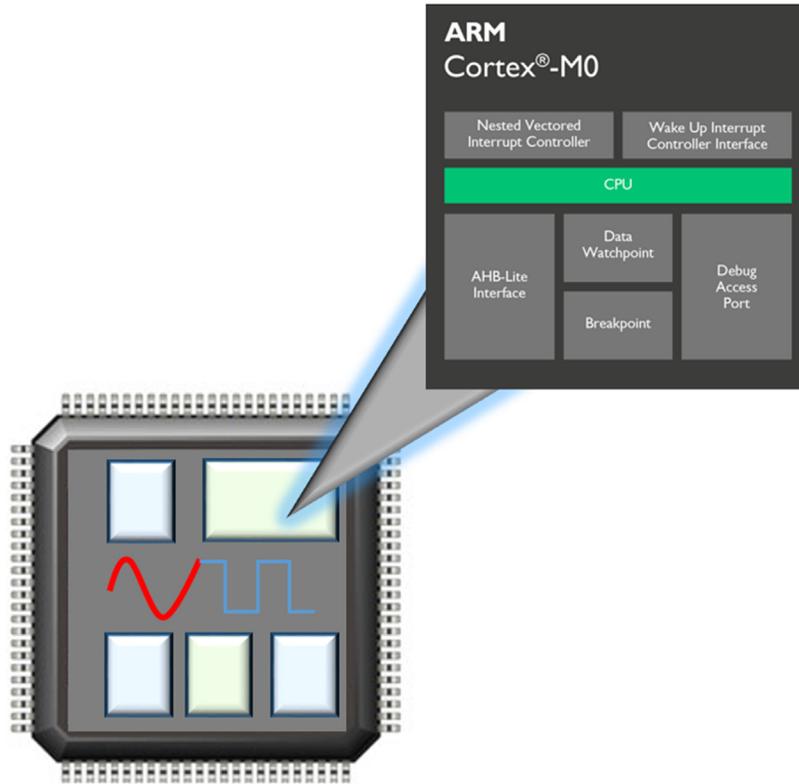


Figure 1: The scenario – expanding the AMS design to include a processor.

Because of the lack of funding, you need to keep non-recurring engineering (NRE) cost as low as possible. The working definition of NRE in the context of this paper is the cost of IP and EDA tooling. Because your salary is recurring, the time spent proving the concept is not included as NRE, but you only have a few days to complete the proof of concept anyway. How can you show a proof-of-concept to your customer fast and for zero NRE?

ARM® DESIGNSTART™

Recognizing that there are special requirements for sensor and mixed-signal companies, as well as startups or small teams creating custom SoCs, ARM offers the [DesignStart](#) portal (Figure 2) that allows designers fast and easy access to a trial selection of ARM products without charge. In addition, Mentor Graphics provides the Tanner EDA design tools for free evaluation and ARM offers approved design partners for SoC development help.

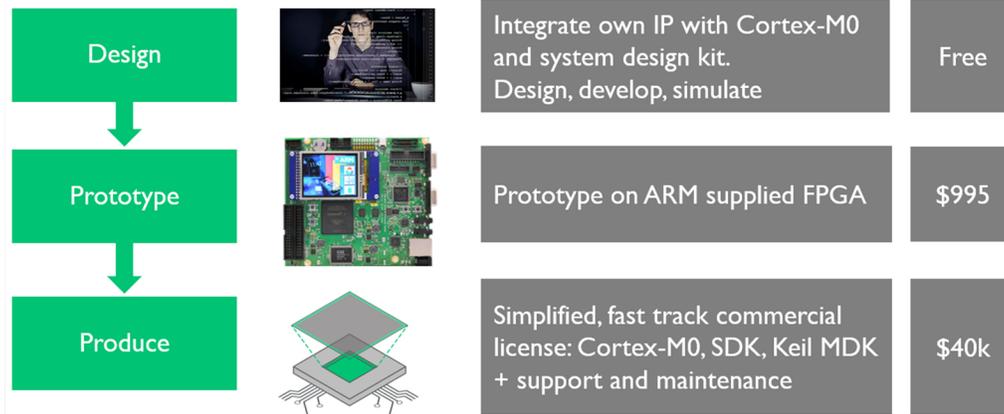


Figure 2: The ARM DesignStart portal (Source: ARM).

For your project, the portal offers the ARM Cortex[®]-M0 processor that you can download and use for design and simulation without charge. This is the ideal solution to your rapid proof-of-concept project. The ARM Cortex-M0 is a low-power 32-bit processor with a small footprint (Figure 3).

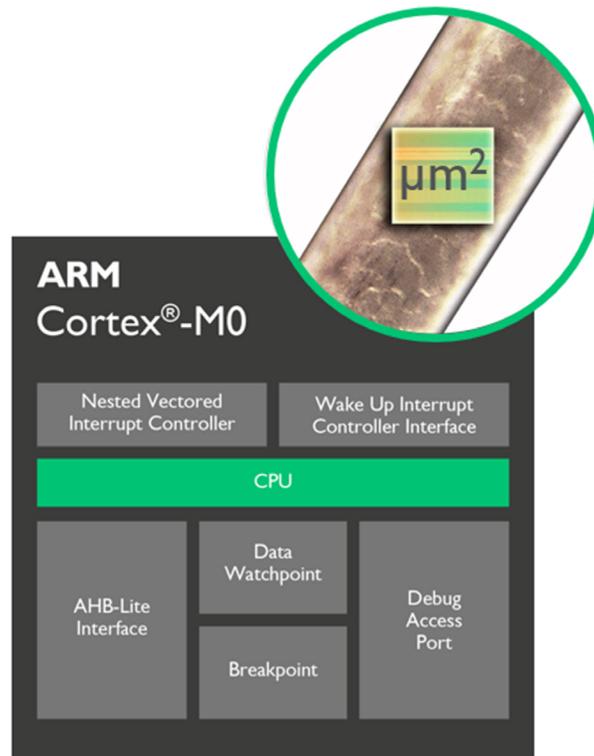


Figure 3: The ARM Cortex-M0 processor (Source: ARM).

This processor is widely-used in the industry for cost-sensitive devices and it has the following key features:

- Built-in low-power features - such as sleep, deep sleep, and state retention low-power modes
- Deterministic instruction execution timing - instructions and interrupts have a fixed timing and interrupt handling is automatic
- Exceptional code density - compact code with smaller code than 8/16-bit devices
- Tiny footprint – only 12k gates, resulting in 32-bit processing with a gate count of an 8-bit processor
- Simple and quick development – with just 56 instructions and one AHB bus interface, it is possible to quickly master the entire Cortex-M0 instruction set and its C-friendly architecture

If you are new to the Cortex-M0 processor, learn more about it on [ARM Developer](#).

TANNER EDA SOLUTION

Mentor graphics offers a 30-day, [free evaluation](#) of the Tanner EDA tools that you can use to design and simulate your proof-of-concept SoC. Tanner EDA provides a complete AMS IC solution in a highly-integrated end-to-end flow. For your proof-of-concept project, you can create the AMS schematic using S-Edit to integrate the ARM Cortex-M0 core and simulate the entire design using T-Spice and ModelSim (Figure 4). After you prove the concept, you can use the full suite of implementation tools to layout and verify the design (not covered in this paper).

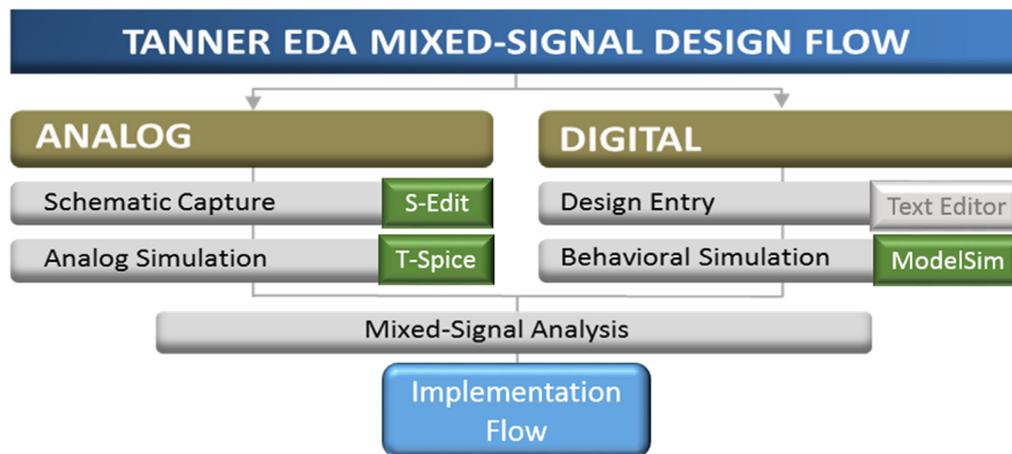


Figure 4: The Tanner design and simulation flow.

To prepare for your project, the next steps are:

1. Register for [DesignStart](#) to gain free access to the Cortex-M0 processor.
2. After approval, download the Cortex-M0 Design Kit to your project area.
3. Register for the free 30-day evaluation of the Tanner EDA tools.
4. Download the Tanner EDA tools and set up your license.

At this point, you are ready to start your proof-of-concept design.

THE SAMPLE DESIGN

The best way to understand the major steps of creating the proof of concept is to work through a sample design (Figure 5).

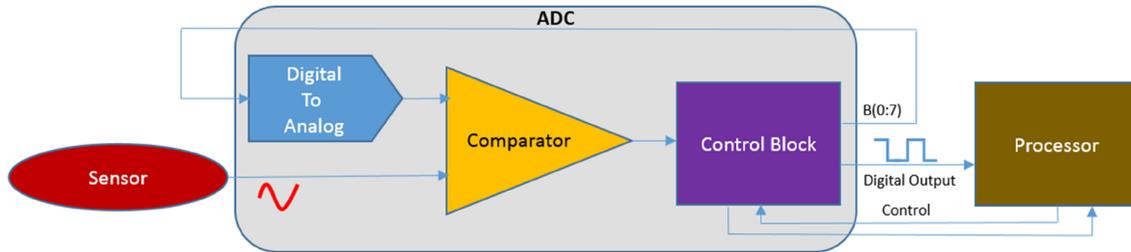


Figure 5: The sample design – a sensor driving an ADC integrated with the processor.

Our sample design shows the connection of an analog sensor to an 8-bit analog to digital converter (ADC) that we want to connect to the ARM Cortex-M0 processor. Because this is a proof-of-concept, you don't need an elaborate software program running on the processor. Instead, you just need to simulate that the digital serial output from the Control Block communicates properly with the processor. So, the focus of the project is on the correct interface of the AMS design to the processor.

THE DESIGN AND SIMULATION FLOW

The ARM Cortex-M0 [DesignStart Design Kit](#) (Figure 6) provides you with a pre-integrated processor subsystem with peripheral components. Using the sample design, you connect the Control Block to the AHB2APB interface, which connects to the AHB interface. The ARM AMBA® Advanced High-Performance Bus (AHB) specification is the main system bus for the processor and the AMBA Advanced Peripheral Bus (APB) specification is for connecting peripheral components. The system consists of Verilog source files with the ARM Cortex-M0 functionality obfuscated from you (but not from the simulator).

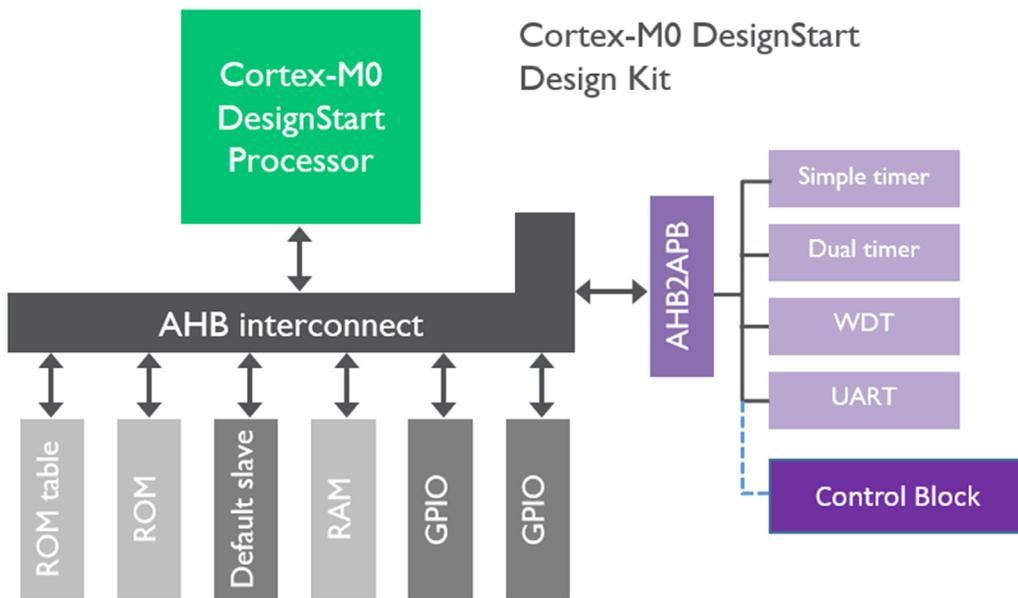


Figure 6: Connecting the control block to the ARM Cortex-M0 processor (Source: ARM).

CREATING THE INTERFACE

The Control Block is a Verilog module (Figure 7) that describes the behavior of the component. You can create this description using the text editor in S-Edit.

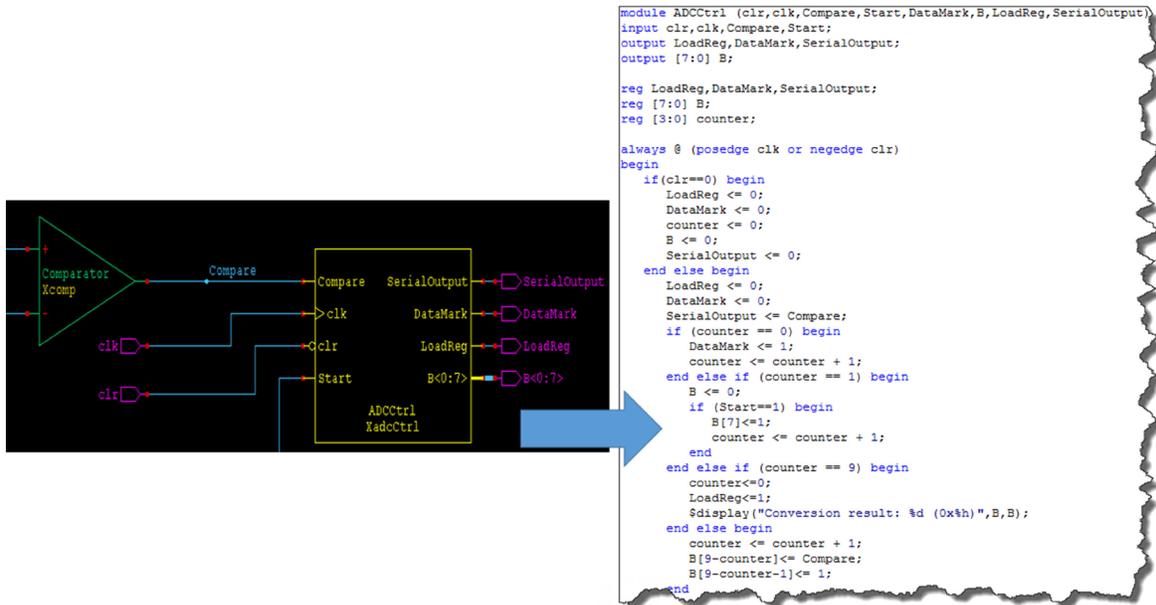


Figure 7: Verilog description of the control block.

In order to connect the Control Block (*ADCCtrl*) to the subsystem bus, you need to create a Verilog module that maps the inputs and outputs of the Control Block to the APB. This means that you need to understand the APB standard document (that you can [download](#) for free from ARM). Within a text file, you specify a module (Figure 8) that:

1. Defines the APB inputs and outputs.
2. Specifies the design inputs and outputs that you need to connect from the Control Block.
3. Specifies any design signals that are necessary for use in the behavior code (Figure 9).
4. Establishes the port mapping between the Control Block, design I/O, and this interface module.

```

module apb_adc8 (
  input wire      PCLK,      // PCLK for timer operation
  input wire      PCLKG,    // Gated clock
  input wire      PRESETn,  // Reset

  input wire      PSEL,     // Device select
  input wire [11:0] PADDR,  // Address
  input wire      PENABLE,  // Transfer control
  input wire      PWRITE,  // Write control
  input wire [31:0] PWDATA, // Write data

  output reg [31:0] PRDATA, // Read data
  output reg      PREADY,  // Device ready
  output reg      PSLVERR, // Device error response

  input wire      ADC_COMPARE,
  output wire [7:0] ADC_DAC_DATA

);

reg Start;
wire DataMark, LoadReg, SerialOutput;

reg idle;

ADCCtrl u_adcctrl (.clr(PRESETn),.clk(PCLK),.Compare
(ADC_COMPARE),.Start(Start),.DataMark(DataMark),.B
(ADC_DAC_DATA),.LoadReg(LoadReg),.SerialOutput(SerialOutput));

always @(posedge PCLK or negedge PRESETn)
begin
  if (!PRESETn)
  begin

```

Figure 8: Defining the interface and port mapping.

Next, you define the actual behavior of the I/O and signals in the same file (Figure 9) including the state of readiness, when to wait, and how to send and receive data.

```

always @(posedge PCLK or negedge PRESETn)
begin
  if(!PRESETn)
  begin
    PRDATA <= {32{1'b0}};
    PSLVERR <= 1'b0;
    Start <= 1'b0;
    idle <= 1'b0;
    PREADY <= 1'b0;
  end
  else
  begin
    Start <= 1'b0;
    PRDATA <= {32{1'b0}};
    if(PENABLE && PSEL && ~DataMark && ~PWRITE && ~idle)
    begin
      Start <= 1'b1;
      idle <= 1'b1;
    end
    if(PENABLE && PSEL && DataMark && ~PWRITE)
    begin
      PRDATA[7:0] <= ADC_DAC_DATA;
      PREADY <= 1'b1;
    end
    if(~PSEL || ~PENABLE)
    begin
      idle <= 1'b0;
      PREADY <= 1'b0;
    end
  end
end
endmodule

```

Figure 9: Bus interface behavior.

PLUG INTO THE MCU SYSTEM

Next, we need to connect our peripheral to the Cortex-M0 system. The MCU system block provides several APB extension ports for this purpose. For this example, we choose port 15. Within the definition section of the Verilog code that represents the system, the signals for the ADC are added (Figure 10).

```

wire [31:0] ADC8_PWDATA, ADC8_PRDATA, ADC8_PADDR;
wire ADC8_PSEL, ADC8_PENABLE, ADC8_PWRITE, ADC8_PREADY, ADC8_PSLVERR;

```

Figure 10: Define ADC signals.

Within the same file, in the configuration section of the APB subsystem, the ADC signals are connected to the signals of the corresponding APB port 15 on the APB. Finally, the APB interface module is instantiated (Figure 11).

```

apb_adc8 u_adc8 (
    .PCLK(PCLK),
    .PCLKG(PCLKG),
    .PRESETn(PRESETn),
    .PSEL(ADC8_PSEL),
    .PADDR(exp_paddr),
    .PENABLE(exp_penable),
    .PWRITE(exp_pwrite),
    .PWDATA(exp_pwdata[31:0]),
    .PRDATA(ADC8_PRDATA),
    .PREADY(ADC8_PREADY),
    .PSLVERR(ADC8_PSLVERR),
    .ADC_COMPARE(adc_compare),
    .ADC_DAC_DATA(adc_dac_data)
);

```

Figure 11: Instantiating the ADC in the APB interface module.

WRITING THE SOFTWARE

After you interface the Control Block to the Cortex-M0 subsystem through the definition of the interface, you need to write a test program that runs on the processor. Because the program must be compiled to target the Cortex-M0, you can use ARM Keil® MDK-Lite which is a software development solution for creating, compiling, and debugging your program. See the [download](#) page for details about getting an evaluation copy. Alternatively, you can use the professional version of Keil MDK with a 90 day license that comes with the [DesignStart package](#).

The easiest way to quickly create a software test program is to copy and modify the *hello.c* file that the DesignStart Design Kit contains (Figure 12).

```

#include <stdio.h>
#include "uart_stdout.h"
#define ADCDATA (* (volatile unsigned char*)0x4000F000L);

int main (void)
{
    // UART init
    UartStdOutInit();
    printf("Hello world\n");

    int v = ADCDATA;
    printf("Reg value is %d\n", v);

    if(v==215)
        printf("*** TEST PASSED **\n");
    else
        printf("*** TEST FAILED **\n");
    // End simulation
    UartEndSimulation();

    return 0;
}

```

Figure 12: Test software program.

The software program exercises the ADC from within the simulated Cortex-M0. *ADCDATA* is defined and set to the base, hex value of the memory-mapped address of the APB port 15. The program executes *printf* statements in the

simulator through the UART module. The ADC input voltage that represents the sensor signal is set to 1.8V. The ADC reference is set to 2.2V. This corresponds to the ADC output value of 215 counts ($(1.85V/2.2V) * 256$). The program checks if 215 counts were simulated. If so, the test passes. If not, the test fails.

Use the Keil MDK-Lite tool to compile this program into the ModelSim *work* directory.

SIMULATING THE DESIGN

S-Edit creates the Verilog-AMS netlist and passes it to T-Spice. T-Spice splits the netlist automatically to partition the design for analog simulation and for digital simulation in ModelSim, as Figure 13 shows.

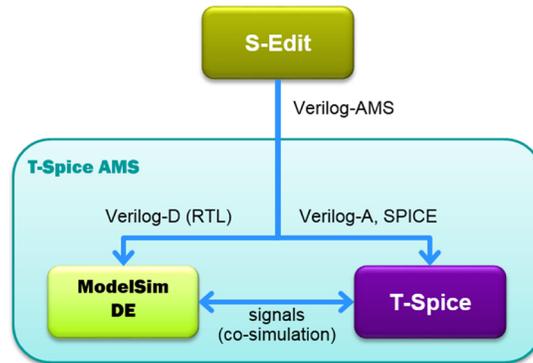


Figure 13: Analog and digital partitions for simulation.

Both simulators invoke automatically, and during simulation, the signal values are passed back and forth between the simulators. This means, that regardless of the design implementation (SPICE or Verilog), you just run the simulation from S-Edit and the design is automatically partitioned across the simulators. Then, you can view the results using the ModelSim or T-Spice waveform viewers.

The sample design contains a testbench that provides the voltage sources and the clock and captures I/O values for waveform display (Figure 14). For the proof-of-concept test, a constant analog voltage of 1.8V is substituted for the sensor signal.

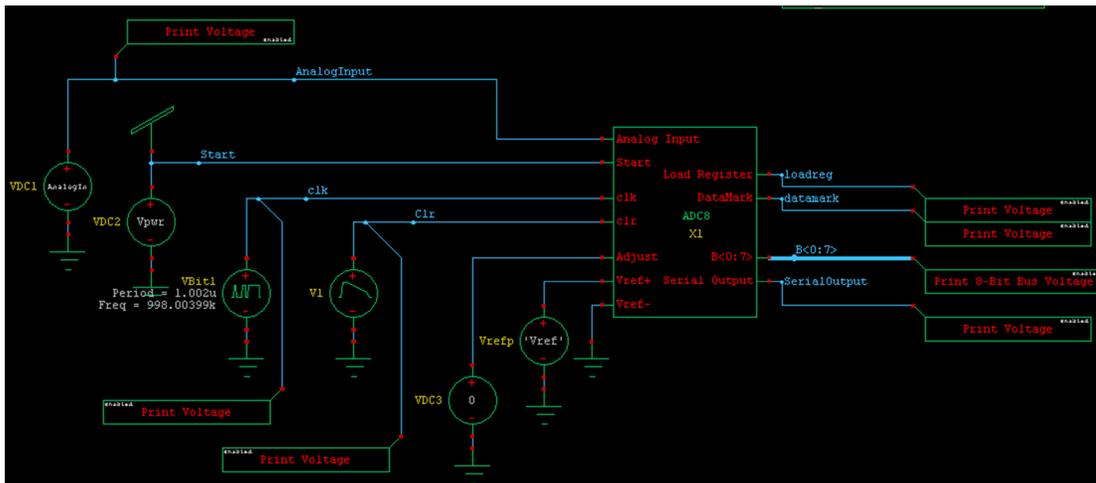


Figure 14: Sample design testbench.

The 8-bit ADC performs a successive approximation to convert the analog signal from the sensor into a discrete digital representation to input into the processor. The comparator contrasts the analog input to the output to the digital to analog (DAC) as the Control Block performs a higher or lower “guess” as the waveforms in Figure 15 show.

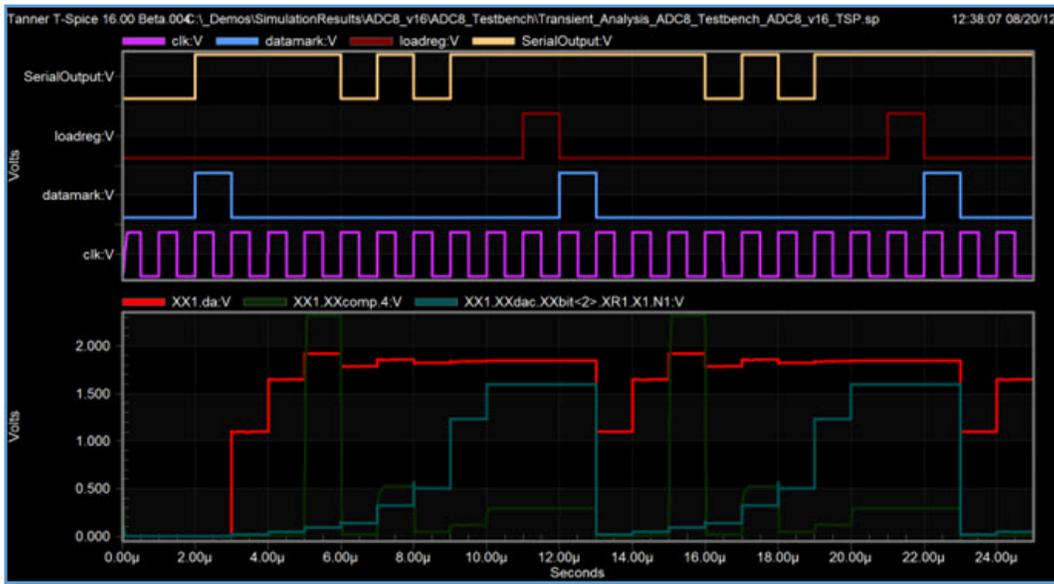


Figure 15: Waveforms showing successive approximation.

In a successful simulation, you would see the print statements from the Control Block and from the software program.

SUMMARY

The proof-of-concept project focuses on accurately interfacing your AMS design to the [Cortex-M0 processor](#) to show your end customer that the design is feasible. You can complete this project quickly while expending zero NRE:

Item	Cost
ARM Cortex-M0 DesignStart Design Kit	\$0
Tanner EDA toolset (30 day evaluation)	\$0
AMBA APB specification	\$0
Keil MDK-Lite software development solution	\$0
Non-NRE: Create & validate the proof-of-concept	Days X pay per day

THE NEXT STEPS

With the proof-of-concept in hand, you can create a demonstration and possibly a few presentation slides in order to describe the project to your customer. After the customer is satisfied, the next step is to implement the SoC. This requires you to purchase the tools in the flow and to purchase the Cortex-M0 and Design Kit with a simplified, fast-track license that is available from ARM. Then, you need to work through the implementation flow (Figure 16) before sending the design to your chosen foundry.

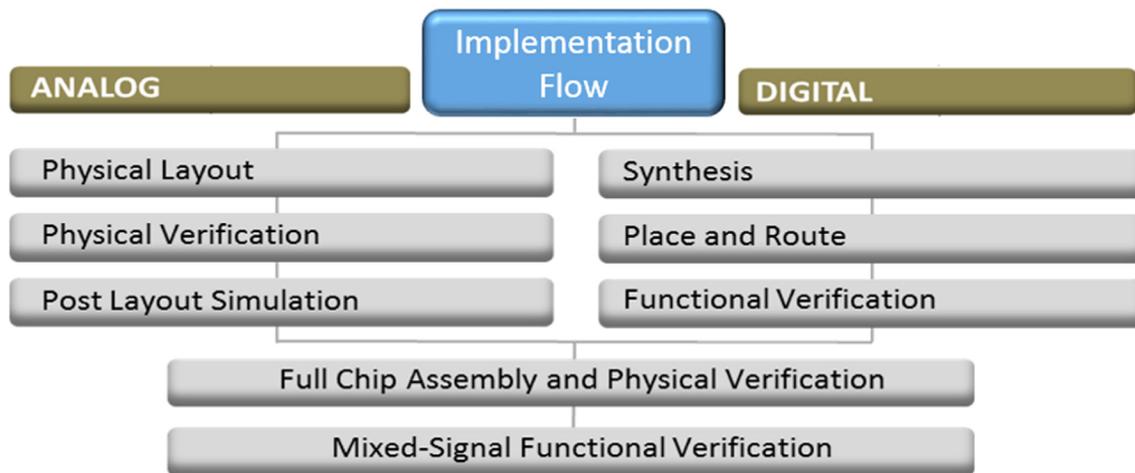


Figure 16: The Tanner EDA implementation flow.



ARM, AMBA, Cortex, DesignStart, Keil, are trademarks or registered trademarks of ARM Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. All other brands or product names are the property of their respective holders.

[Further detail.](#)

For the latest product information, call us or visit:

www.mentor.com

©2017 Mentor Graphics Corporation, all rights reserved. This document contains information that is proprietary to Mentor Graphics Corporation and may be duplicated in whole or in part by the original recipient for internal business purposes only, provided that this entire notice appears in all copies. In accepting this document, the recipient agrees to make every reasonable effort to prevent unauthorized use of this information. All trademarks mentioned in this document are the trademarks of their respective owners.

Corporate Headquarters
Mentor Graphics Corporation
 8005 SW Boeckman Road
 Wilsonville, OR 97070-7777
 Phone: 503.685.7000
 Fax: 503.685.1204

Sales and Product Information
 Phone: 800.547.3000
sales_info@mentor.com

Silicon Valley
Mentor Graphics Corporation
 46871 Bayside Parkway
 Fremont, CA 94538 USA
 Phone: 510.354.7400
 Fax: 510.354.7467

North American Support Center
 Phone: 800.547.4303

Europe
Mentor Graphics
 Deutschland GmbH
 Arnulfstrasse 201
 80634 Munich
 Germany
 Phone: +49.89.57096.0
 Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics (Taiwan)
 11F, No. 120, Section 2,
 Gongdao 5th Road
 HsinChu City 300,
 Taiwan, ROC
 Phone: 886.3.513.1000
 Fax: 886.3.573.4734

Japan
Mentor Graphics Japan Co., Ltd.
 Gotenyama Trust Tower
 7-35, Kita-Shinagawa 4-chome
 Shinagawa-Ku, Tokyo 140-0001
 Japan
 Phone: +81.3.5488.3033
 Fax: +81.3.5488.3004

