Review Article

International Journal of Advanced Computer Research, Vol 6(26)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.2016.625018

State of the art: benchmarking microprocessors for embedded automotive applications

Adnan Shaout* **and Anthony Walker**
Department of Electronics and Communication Engineering, University of Michigan, Dearborn, United States

Received: 11-June-2016; Revised: 11-August-2016; Accepted: 16-August-2016

Abstract
Benchmarking microprocessors provides a way for consumers to evaluate the performance of the processors. This is done by using either synthetic or real world applications. There are a number of benchmarks that exist today to assist consumers in evaluating the vast number of microprocessors that are available in the market. In this paper an investigation of the various benchmarks available for evaluating microprocessors for embedded automotive applications will be performed. We will provide an overview of the following benchmarks: Whetstone, Dhrystone, Linpack, standard performance evaluation corporation (SPEC) CPU2006, embedded microprocessor benchmark consortium (EEMBC) AutoBench and MiBench. A comparison of existing benchmarks will be given based on relevant characteristics of automotive applications which will give the proper recommendation when benchmarking processors for automotive applications.

Keywords
Benchmarking, Embedded systems, Automotive applications, Microprocessors, Synthetic benchmark.

1.Introduction
Benchmarking microprocessors provides a way for consumers to evaluate the performance of the processors. This is done by using either synthetic or real world applications. In some cases, these real world applications tend to be software packages that are used by consumers in production intent environments (commonly referred to as proprietary software). Synthetic benchmarks are those that simulate large programs and/or real world applications. An aspect of real world applications is the use of various mathematical computations (integer math, floating point math, infinite impulse response (IIR) filters, etc.).

While looking for articles concerning the methods for benchmarking microprocessors for embedded automotive applications, we found it difficult to find information that focused solely on benchmarking microprocessors for embedded automotive applications. One reason is that there are not many benchmarks available are due to the fact that EEMBC consortium is comprised of all the main suppliers of microprocessors in the automotive industry. These companies include Freescale, Infineon, ST and Renesas to name a few [1]. There seemed to be various methodologies and techniques of benchmarking microprocessors ranging from desktop computers and cell phones to those used for telecommunications.

2.Overview of benchmarks
The Whetstone benchmark is a synthetic benchmark that was written in 1972 by Dr. B.A. Wichman and Harold Curnow [2]. Wichman created a set of 42 simple statements using the algorithmic language 1960 (ALGOL 60) programming language. In addition to being written in ALGOL 60 it was also later written in Pascal, formula translation (FORTRAN) and C [3, 4]. This benchmark was created to measure the speed and efficiency of a computer that performs floating-point operations [5]. It is comprised of a variety of functions including sin, cos, square root, exponents, logarithmic operations, integer and floating point operations. Its name is derived from the compiler system used to collect statistics about the distribution of Whetstone instructions (the Whetstone Algol compiler system) [3]. Some important characteristics of the Whetstone benchmark are as follows [3]:

*Author for correspondence

- It contains a high percentage floating-point data and floating point calculations.
- A high percentage of the execution time is spent in mathematical library functions.
- The use of local variables is very limited.
- Global variables are heavily used. (NOTE: The use of global variables is not recommended in embedded applications due to coupling and increased complexity).
- Due to its construction principle of using 9 small loops, it has an extremely high code locality.

The Dhrystone benchmark is also a synthetic benchmark. It was written in 1964 by Reinhold Weicker [5, 6]. Dhrystone was originally written in analysis, design and algorithm (ADA) and was designed in a way that was intended to make it possible to develop in other programming languages. An instance of this is Pascal. This was a relatively easy translation due to the fact that Dhrystone uses a "Pascal subset" of ADA [6]. The other programming language used to implement the Dhrystone benchmark is C. However, using C posed a number of challenges and hence yields other possible versions [6] such as a version without register variables, a version that declares every local variable of a scalar type to be a register variable and a version where the programmer optimizes carefully, trading off the benefit of register variables in terms of access time against the additional overhead in procedure call and return.

The Dhrystone benchmark is used to measure and compare the performance of different computers [7]. It concentrates on string handling and does not use floating point arithmetic [8]. As stated by the ARM Keil website "it is heavily influenced by hardware and software design, compiler and linker options, code optimizing, cache memory, wait states, and integer data types" [8]. This was also reiterated by Walter J. Price, who stated that the Dhrystone "benchmark measures processor and compiler efficiency by executing a typical set of integer calculations. These calculations include integer arithmetic, character/string/array manipulation, and pointers." [4]. Some important characteristics of the Dhrystone benchmark are [3] no floating point operations in its measurement loop, a sizable percentage of execution time is spent in string functions, it contains hardly any loops within the main measurement loop, a small amount of global data is manipulated and no attempt is made to prevent compiler optimizations. The Linpack benchmark created by Jack Dongarra, Jim Bunch, Cleve Moler and Gilbert Stewart and published in 1976 was not originally a benchmark [3]. It has been a collection of linear algebra subroutines often used in FORTRAN programs that emphasized floating point addition and multiplication [3, 4]. These subroutines were referred to as basic linear algebra subroutines (BLASs). They came in two forms: Coded and FORTRAN. The Coded BLASs were written in assembly language while the Fortran BLASs were written in FORTRAN. Important characteristics of the Linpack benchmark are as follows [3]:

- A high percentage of floating-point operations are performed.
- No mathematical functions are used.
- Execution time is almost spent exclusively in one small function.
- High code locality and low data locality.

The SPEC CPU2006 benchmark is the SPEC benchmark suite. According the SPEC website it is their "next-generation, industry standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler" [7]. There are 2 components of this benchmark suite: integer and floating point. The integer suite (SPECint 2006) contains 12 benchmark tests (described in *Table 1* [9]). The floating point suite (SPECfp 2006) contains 19 benchmark tests (described in *Table 2* [10]).

Table 1 Description of integer SPECint 2006 benchmark tests

Benchmark	Programming language	Application area	Brief description
400.perlbench	C	Programming Language	Derived from Perl v5.8.7. The workload includes SpamAssasin, MHonArc (an email indexer) and specdiff (SPEC's tool that checks benchmark outputs).
401.bzip2	C	Compression	Julian Seward's bzip2 version 1.0.3. Modified to do most work in memory rather than doing I/O.

Benchmark	Programming language	Application area	Brief description
403.gcc	C	C Compiler	Based on GCC v3.2. Generates code for Opteron microprocessor.
439.mcf	C	Combinatorial Optimization	Vehicle scheduling. Uses a network simplex algorithm (which is also used in commercial products) to schedule public transport.
445.gobmk	C	Artificial Intelligence	Plays the game of Go (a simply described but deeply complex game).
456.hmmer	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models (HMMs).
458.sjeng	C	Artificial Intelligence	A highly-ranked chess program that also plays several chess variants.
462.libquantum	C	Physics/Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
464.h264ref	C	Video Compression	A reference implementation of H.264/AVC. Encodes a video stream using 2 parameter sets. The H.264/AVC standard is expected to replace MPEG2.
471.omnetpp	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
473.astar	C++	Path-finding Algorithms	Pathfinding library for 2D maps, including the well know A* algorithm.
483.xalancbmk	C++	XML Processing	A modified version of Xalan-C++ which transforms XML documents to other document types.

Table 2 Description of floating point SPECfp 2006 benchmark tests

Benchmark	Programming language	Application area	Brief description
410.bwaves	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow.
416.gamess	Fortran	Quantum Chemistry	Implements a wide range of quantum chemical computations.
433.milc	C	Physics/Quantun Chromodynamics	A gauge field generating program for lattice gauge theory programs with dynamical quarks.
434.zeusmp	Fortran	Physics/CFD	A computational fluid dynamics code developed at the Laboratory for Computational Astrophysics for the simulation of astrophysical phenomena.
435.gromacs	C, Fortran	Biochemistry/Molecular Dynamics	Molecular dynamics i.e. simulate Newtonian

Benchmark	Programming language	Application area	Brief description
			equations of motion for hundreds to millions of particles.
436.cactusADM	C, Fortran	Physics/General Relativity	Solves the Einstein evolution equations using a staggered-leapfrog numerical method.
437.leslie3d	Fortran	Fluid Dynamics	Computational Fluid Dynamics using Large-Eddy Simulations with Linear-Eddy Modil in 3D.
444.namd	C++	Biology/Molecular Dynamics	Simulates large biomolecular systems.
447.dealII	C++	Finite Element Analysis	A C++ program library targeted at adaptive finite elements and error estimation.
450.soplex	C++	Linear Programming, Optimization	Solves a linear program using a simplex algorithm and sparse linear algebra.
453.povray	C++	Image Ray-tracing	Image rendering.
454.calculix	C, Fortran	Structural Mechanics	Finite element code for linear and nonlinear 3D structural applications.
459.GemsFDTD	Fortran	Computational Electromagnetics	Solves the Maxwell equations in 3D using the finite-difference time-domain (FDTD) method.
465.tonto	Fortran	Quantum Chemistry	An open source quantum chemistry package using an object-oriented design in Fortran 95 that places a constraint on a molecular Hartree-Fock wave function calculation to better match experimental X-ray diffraction data.
470.lbm	C	Fluid Dynamics	Implements the "Lattice-Boltzmann Method" to simulate incompressible fluids in 3D.
481.wrf	C, Fortran	Weather	Weather modeling from the scales of meters to thousands of kilometres.
482.sphinx3	C	Speech recognition	A widely-known speech recognition system from Carnegie-Mellon University.

The Autobench benchmark is the one provided by the EEMBC. This consortium was created in April 1997 to develop meaningful performance benchmarks for processors in embedded applications [11]. Markus Levy, the founder of EEMBC, created a set of benchmarks that would provide better information in the analysis of microprocessors, microcontrollers and compilers to address the ineffectiveness of Dhrystone as a tool for evaluating embedded processor performance [12]. EEMBC offers a variety of benchmark suites to evaluate processors for various types of applications. Some examples of these benchmarks are shown in *Table 3* [13].

Table 3 Example benchmarks from the EEMBC benchmark suites [13]

Suite	Benchmarks
Automotive	Finite and infinite impulse response (FIR and IIR) filters, tooth-to-spark tests, pulse-width modulation, matrix multiplication and shifting, table lookup and, fast Fourier transform (FFT).
Consumer	JPEG compression and decompression, high pass grayscale filter, RGB to CMYK, and RGB to YIQ converter.
Digital Entertainment	JPEG compression and decompression, high pass grayscale filter, RGB to CMYK, and RGB to YIQ converter, advanced encryption standard (AES), and data encryption standard (DES).
Networking	Packet flow algorithms, open shortest path first (OSPF), and route lookup.
Networking v2.0	Packet check algorithms, OSPF, RSA, and network address translator (NAT).
Office automation	Dithering, rotate, and text.
Telecommunications	Autocorrelation, FFT, and Viterbi decoder.

Since the focus of this research is within the automotive domain, only the suites that can potentially impact the evaluation of processors and compilers are discussed.

The automotive benchmark, as of June 2004, is comprised of the following algorithms [14] (a brief description of each algorithm as expressed by EEMBC will be given):

- Angle-to-time conversion
- Basic floating point
- Bit manipulation
- Cache buster
- Controller area network (CAN) remote data request
- FFT
- FIR
- IIR
- Inverse discrete cosine transform (IDCT)
- Matrix arithmetic
- Pointer chasing
- Pulse width modulation
- Road speed calculation
- Table lookup and interpolation
- Tooth-to-spark

Angle-to-time conversion algorithm simulates the crankshaft of an engine by reading a counter which measures the real-time delay between pulses sensed from the gear on it [15]. This applies to embedded automotive applications such as the engine control module. Basic floating point math is used in applications such as Powertrain, anti-lock brake system (ABS), traction control and active suspension [16]. The FFT takes any function and converts it to an equivalent set of sine waves [17]. FFT is used in digital signal processing. Bit manipulation is the act of algorithmically manipulating bits or other pieces of data shorter than a word [18]. This is used highly in embedded software applications. A few of examples of bit manipulation are given as follows:

- Masking out the upper nibble of a byte to obtain the lower nibble only.
- Masking out the lower nibble of a byte and shifting it right by 4 to obtain the upper nibble only.
- Setting a bit to indicate a fault is present in a system
- Clearing a bit to indicate a fault is no longer present in a system.

In order to simulate microprocessors that do not have a cache, EEMBC has implemented the cache buster benchmark. As stated by Markus Levy, it is used to highlight scenarios where long sections of control code are executed with very little branching or use of the same data [19].

The CAN remote data request benchmark simulates the scenario where a remote data request is received by all nodes on the bus. Upon receiving the request, each node parses the identifier determines if they are responsible for responding to the request. Once a node has determined it must respond to the request, the data is gathered to be transmitted on the bus to the node that made the request [20].

The FFT algorithm emulates an application "performing a power spectrum analysis of a time varying input waveform" [21]. This is done by getting the test data and input values, running the FT calculation and then calculating the power spectrum.

To simulate applications where an FIR filter is used for fixed point values, the FIR benchmark is used. This algorithm gets the input test data and values, calculates the FIR low pass filter and then the FIR high pass filter [22]. IIR filters are used for filtering data samples that are fixed-point values. The algorithm employed by EEMBC uses a Direct Form II N-Cascaded Direct second order IIR filter [23]. It characterizes the microprocessors ability to perform multiple accumulates and rounding [24]. The Pointer Chasing algorithm uses doubly linked lists to exercise an application/program that utilizes pointer arithmetic. Pointer arithmetic was considered a violation of the MISRA C: 2004 coding rules. However, the MISRA C: 2012 coding rules now make pointer arithmetic an advisory rule [24]. Pointers have been used extensively in embedded automotive applications for some time now. By updating this standard, the MISRA C guidelines seem to have caught up with the industry.

The IDCT is used to reconstruct a sequence of coefficients from the discrete cosine transform (DCT) [25]. This sequence of coefficients can be derived from a picture or video file. *Figure 1* shows an outline of a typical image/video transmission [26].

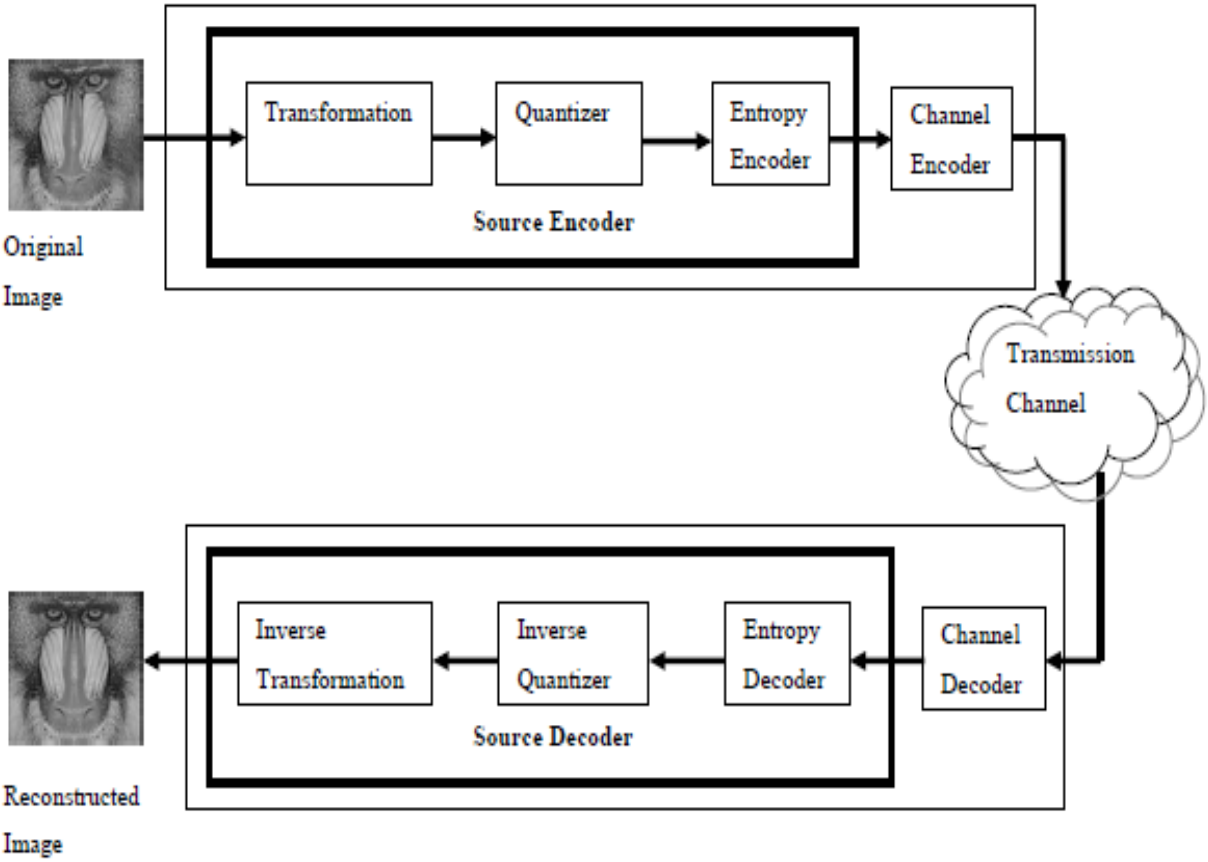

Figure 1 An outline of a typical image/video transmission [26]

One way that this transform is potentially used in automotive applications is to reconstruct data that is received from a rear facing camera. This data would go through the source encoder and be sent over a transmission channel (Flex-Ray, CAN or LIN) to a video display for vehicles equipped with a rear-view camera. The matrix arithmetic algorithm is used to simulate applications that perform a significant amount of matrix algebra. This algorithm uses lower upper (LU) decomposition on 'n x n' input matrices, computes the determinant of the input matrix and a cross product with a second matrix [27].

The pulse-width modulation (PWM) algorithm simulates a scenario where an actuator (motor) is controlled by a PWM signal proportional to an input

[2]. This algorithm presumes the processor is driving a motor driver with both direction and enables signals [2]. The final benchmark is MiBench. MiBench follows an EEMBC's model of benchmark suites. Matthew Guthaus, Jeffrey Ringenberg Dan Ernst, Todd Austin, Trevor Mudge and Richard Brown are the creators of this free commercially representative benchmark suite. It is comprised of 35 applications that are divided into the following six suites [28]:

- Automotive and industrial control
- Consumer devices
- Office automation
- Network
- Security
- Telecommunications

The MiBench automotive benchmark suite is slightly different than EEMBC's. The tests (or algorithms) used here are as follows:

- Basic math
- Bit counting
- Sorting
- Shape recognition

The basic math test (Basicmath) performs calculations such as a cubic function solving, integer square root and angle conversions from degrees to radians for calculating road speed [28] (or other vector values). For example, the speed of a vehicle can be communicated to electronic control units (ECUs) on the CAN bus. Based on the customer requirements, the ECU that receives the speed message will have to do some basic math operations on it. This could be due to the fact that each bit of the byte representing the speed accounts for a certain speed (i.e. 1 count equals 0.234 kph). Floating point arithmetic could also come into play here if decimal numbers are used.

The bit count algorithm (Bitcount) is used to test the microprocessor's ability to manipulate bits by counting the number of bits in an array of integers. There are 5 different algorithms used for these tests that are 1) an optimized 1-bit per loop counter, 2) recursive bit count by nibbles, 3) non-recursive bit count by nibbles using a table look-up, 4) non-recursive bit count of bytes using a table look-up and 5) shift and count bits [28]. The sorting algorithm (qsort) is the well-known quick sort algorithm that many have used at some point in their academic and/or professional careers. There are two data sets used for the testing. A large data set composed of three-tuples representing points of data and a small data set that is a list of words [28].

The shape recognition algorithm was developed for recognizing corners and edges in magnetic resonance images of the brain [28]. The algorithm can smooth the image. It also has spatial control. There are two data sets used for testing. A large data set that contains a complex picture; and a small data set that contains a black and white image of a rectangle.

As MiBench is modeled after EEMBC, the FFT/ inverse fast Fourier transform (IFFT) test in the Telecommunications suite is also in EEMBC's automotive suite. The FFT and IFFT perform on an array of data that is a polynomial function with pseudorandom amplitude and frequency sinusoidal components. One additional test in the Telecommunications suite that can be applied to the automotive domain is the cyclic redundancy check (CRC) 32 test. This test is used to detect errors in data transmission.

All of these aspects simulate real world applications. Using the EEMBC suites reduces the time the consumer is required to dedicate to selecting their microprocessors. Consumers can select devices based on their score from EEMBC and use their software that contains their Intellectual Property to make a final decision. Of course the other option is to have the microprocessor vendors come to you with their devices in order to perform your own analysis using proprietary software.

3.Comparison of the benchmarks

Since the focus of this paper is on benchmarking processors for automotive embedded applications, the benchmarks will be compared with respect to the following features:

- The cost associated with its use
- The use of floating point math
- The use of integer math
- If it is available using C programming language
- Whether or not it is meant for automotive applications

These features were chosen since they are the most relevant to automotive embedded applications. *Table 4* shows a comparison of the benchmarks used for automotive embedded applications with respect to the features above.

Table 4 Comparison of benchmarks

	Available using C	Fee Required	Uses integer math	Uses floating point math	Geared towards automotive applications
Benchmark					
Whetstone			X	X	
Dhrystone	X		X		
Linpack				X	
SPEC CPU2006	X	X	X		
Autobench	X	X	X	X	X
MiBench	X		X	X	X

Other characteristics (features) that could be compared are as follows:

- Memory required
- Instruction efficiency
- Speed

These aspects are directly related to the compiler that is used for the software that is being used for the benchmark. Due to this, these characteristics are beyond the scope of this research. Based on the comparison in *Table 4*, Autobench and MiBench have a suite/software package that is geared more towards automotive applications. Although the other benchmarks don't necessarily target automotive applications specifically, there are aspects of each that could be applied. For instance, automotive applications use integer and floating point math to some extent. However, given that a lot of programming is done using C, Whetstone and Linpack would not be viable options for benchmarking for automotive applications.

4.Conclusion

Based on the research and investigation of the benchmarks available today for characterizing microprocessors intended for automotive applications, the EEMBC benchmark is clearly the industry leader. The Whetstone and Dhrystone benchmarks are outdated since they were developed during a time when microprocessors were not as advanced as they are today. The SPEC CPU2006 benchmark, like Autobench, required a fee to obtain the software. The Automotive Suite offered by EEMBC covers every aspect of how microprocessors are used in automotive applications. The fact that all the major suppliers are part of the consortium speaks volumes in and of itself. Consumers can request that a particular microprocessor be evaluated by EEMBC, since the EEMBC Automotive benchmark suite cannot be individually obtained through licensing. Upon completion of the evaluation, the score would be posted on the EEMBC website. It is imperative that the consumer performs their own analysis of a microprocessor performance. During this analysis, the consumer can determine how the processor performs using their proprietary software packages. Aside from the general performance of these benchmarks, there are other aspects that need to be taken into consideration as well. These aspects are as follows:

- Compiler optimizations
- Hardware optimizations
- Architecture of the microprocessor
- Autonomous vehicles
- The use of the cloud
- The Internet of Things

These are all points that needs to be considered and analyzed as part of the process in selecting a microprocessor. Using one of these benchmarks only provides a consumer with a reference point to compare microprocessors. Based on our experience in benchmarking microprocessors, the consumer will still need to perform their own benchmark analysis to get a proper evaluation of a microprocessor performance using their proprietary software. This is further confirmed by the statement in the article *In More Depth: Synthetic Benchmarks* that states "…...no user would ever run a synthetic benchmark as an application because these programs don't compute anything a user would find remotely interesting" [29]. Given the current state of the art to benchmarking microprocessors, we don't see any additional advances that can be made to benchmarking at this time. The performance of the processor itself is only one aspect that needs to be

evaluated. The architecture of the microprocessor along with its hardware optimizations and the optimizations of the compiler being used must be considered as well. The evaluations of these aspects are topics that are covered in other research articles that are not meant to be covered in this paper. While there are advancements being made in the automotive industry as a whole, it is our opinion that the ultimate determining factor for selecting a microprocessor is for the company to develop a proprietary set of benchmarks that can be used across microprocessors and compilers. This can be the biggest asset to selecting the best microprocessor for their given application.

Acknowledgment
None.

Conflicts of interest
The authors have no conflicts of interest to declare.

References
[1] EEMBC Member List. http://www.eembc.org/memberinfo/memberlist.php. Accessed 21 May 2016.
[2] Curnow HJ, Wichmann BA. A synthetic benchmark. The Computer Journal. 1976; 19(1):43-9.
[3] Weicker RP. An overview of common benchmarks. Computer. 1990; 23(12):65-75.
[4] Price WJ. A benchmark tutorial. IEEE Micro. 1989; 9(5):28-43.
[5] Whetstones. http://www.keil.com/benchmarks/whetstone.asp. Accessed 21 May 2016.
[6] Weicker RP. Dhrystone: a synthetic systems programming benchmark. Communications of the ACM. 1984; 27(10):1013-30.
[7] SPEC CPU 2006. https://www.spec.org/cpu2006/. Accessed 15 May 2016.
[8] Dhrystones. http://www.keil.com/benchmarks/dhrystone.asp. Accessed 18 May 2016.
[9] Integer Component of SPEC CPU2006. https://www.spec.org/cpu2006/CINT2006/. Accessed 21 May 2016.
[10] Floating Point Component of SPEC CPU2006. https://www.spec.org/cpu2006/CFP2006/. Accessed 18 April 2016.
[11] Weiss AR. The standardization of embedded benchmarking: Pitfalls and opportunities. In international conference on computer design (ICCD'99) 1999 (pp. 492-508). IEEE.
[12] About EEMBC. http://www.eembc.org/about/index.php. Accessed 21 May 2016.
[13] Poovey JA, Conte TM, Levy M, Gal-On S. A benchmark characterization of the EEMBC benchmark suite. IEEE Micro. 2009; 29(5):18-29.
[14] EEMBC's Automotive /Industrial Microprocessor Benchmarks. http://www.eembc.org/techlit/datasheets/AutomotivePresentation.pdf. Accessed 21 May 2016.
[15] Benchmark Name: Angle to Time Conversion. https://www.eembc.org/techlit/datasheets/auto_angle.pdf. Accessed 11 March 2016.
[16] https://www.arm.com/products/processors/technologies/vector-floating-point.php. Accessed 17 May 2016.
[17] http://www.eembc.org/benchmark/pdf/FPMarkIntroduction.pdf. Accessed 20 April 2016.
[18] Warren HS. Hacker's delight. Pearson Education; 2013.
[19] Cache "Buster". http://www.eembc.org/techlit/datasheets/auto_cache.pdf. Accessed 20 April 2016.
[20] CAN Remote Data Request. http://www.eembc.org/techlit/datasheets/auto_can.pdf. Accessed 21 June 2016.
[21] Fast Fourier Transform, http://www.eembc.org/techlit/datasheets/auto_fft.pdf. Accessed 20 April 2016.
[22] FIR Filter. http://www.eembc.org/techlit/datasheets/auto_fir.pdf. Accessed 20 May 2016.
[23] Cherukuri R, Ryu G. MISRA is now better and easier to implement with polyspace tools. http://www.mathworks.com/products/polyspace/static-analysis-notes/misra-is-now-better-and-easier-to-implement-with-polyspace-tools.html. Accessed 20 May 2016.
[24] IIR Filter. http://www.eembc.org/techlit/datasheets/auto_iir.pdf. Accessed 20 April 2016.
[25] The DCT/IDCT Solution Customer Tutorial. http://homepages.cae.wisc.edu/~ece554/website/Xilinx/app_notes/DCT_IDCT%20Customer%20Tutorial%20custdct.pdf. Accessed 20 April 2016.
[26] The Discrete Cosine Transform (DCT): Theory and Application. http://wisnet.seecs.nust.edu.pk/publications/tech_reports/DCT_TR802.pdf. Accessed 09 May 2015.
[27] Matrix arithmetic. http://www.eembc.org/techlit/datasheets/auto_matrix.pdf. Accessed 21 October 2015.
[28] Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB. MiBench: a free, commercially representative embedded benchmark suite. In IEEE international workshop on workload characterization 2001 (pp. 3-14). IEEE.
[29] In More Depth: Synthetic Benchmarks. http://mprc.pku.edu.cn/courses/organization/autumn2012/hw/INMOREDEPTH/IMD4-SYNTHETIC-BENCHMARKS.PDF. Accessed 30 September 2015.

Dr. Adnan Shaout is a full professor and a Fulbright Scholar in the Electrical and Computer Engineering Department at the University of Michigan – Dearborn. At present, he teaches courses in logic design, computer architecture, cloud computing, fuzzy logic and engineering applications and computer engineering (hardware and software). His current research is in applications of software engineering methods, computer architecture, embedded systems, fuzzy systems, real time systems and artificial intelligence. Dr. Shaout has more than 33 years of experience in teaching and conducting research in the electrical and computer engineering fields at Syracuse University and the University of Michigan - Dearborn. Dr. Shaout has published over 195 papers in topics related to electrical and computer engineering fields. Dr. Shaout has obtained his B.S.c, M.S. and Ph.D. in Computer Engineering from Syracuse University, Syracuse, NY, in 1982, 1983, 1987, respectively.
Email: shaout@umich.edu

Anthony Walker-Engineering Supervisor - ZF TRW from May 2013-Present in Farmington Hills, MI. His main responsibilities are the following: manage software personnel and activities for platform airbag development (6-12 team members globally), track metrics for software platform development, plan software development tasks to meet platform and customer goals, coordinate development activities with global software management team, work with the cross functional global management team to plan/coordinate development activities, provide management support and escalation path for critical issues identified with platform software development, ensure use of approved software methods and procedures, and ensure staff performance goals are met. Principal Product Engineer-Software at TRW from August 2010 – May 2013 in Farmington Hills, MI. He has Master of Science (M.S.) in Software Engineering from the University of Michigan-Dearborn (2015), Bachelor of Science (B.S.) in Computer Engineering from the Michigan State University (2004).