# PCET: A TOOL FOR RAPIDLY ESTIMATING STATISTICS OF WAVEFORM COMPONENTS IMPLEMENTED ON DIGITAL SIGNAL PROCESSORS

James Neel (Cognitive Radio Technologies, LLC, Lynchburg, VA, USA; james.neel@crtwireless.com); Shareef Sayed, Matthew Carrick, Carl Dietrich (Wireless @ Virginia Tech, Blacksburg, VA USA); Jeffrey Reed (Wireless @ Virginia Tech & Cognitive Radio Technologies, LLC)

## ABSTRACT

The Processor Cycle Estimation Tool (PCET) is an extensible open-source tool for rapidly estimating the cycles, power, and memory requirements of waveform components across disparate DSP architectures. This paper reviews the methodologies employed in PCET and compares estimated values with vendor-provided library code and from measured implementations.

## 1. INTRODUCTION

Early in the process of designing an SDR or porting waveforms to new platforms, a systems analysis should be conducted to:

- Identify candidate processing hardware solutions and/or assess the feasibility of porting existing waveforms onto a new platform
- Assess the size, weight, and power (SWAP) impact of implementing the waveforms on the targeted hardware
- Predict the implementation impact of varying critical component parameters (e.g., filter or FFT lengths, code rates)
- Optimize the choice of processors (new design) or the partitioning of existing waveforms across an existing platform's processors (porting), typically by minimizing power, area, memory requirements, and cost (perhaps with varying weights depending on the application).

These analyses require estimates of the waveform's required cycles, execution time, power consumption, and memory usage. However, making reasonably accurate estimates of these parameters is generally a very time-intensive process, a process which is made even more complicated because:

- An SDR has to support numerous waveforms
- There is significant variation in processor architectures.

Unfortunately, existing methods to address these goals do not provide the timeliness, accuracy, and extensibility we would prefer. For example, since most applications for which execution time and power consumption are primary considerations generally utilize hand-coded assembly for their most critical processes, the most accurate method would be to write assembly for the waveform stack for each of the considered processors, but this has prohibitive engineering costs. High-level code (e.g., C-code) could be written and compiled using each of the considered DSP's compilers. However, there is typically such a large variance in performance between compiled high-level code and hand-coded assembly that this method is not generally a satisfactory predictor. Others (e.g., BDTI) provide ratings of processors and most chip vendors will provide profiled libraries of waveform components, but these provide no information if the waveform component of interest was not profiled or rated. As most of the value in an SDR is, by definition, in the software, SDR vendors frequently utilize unique software components for which there is no freely available third party profiling. Thus third-party profiling will generally only have limited applicability.

What is needed is a relatively simple way to estimate the required cycles, execution time, energy, program memory, and data memory for arbitrary waveform components across arbitrary DSP architectures. This solution should be:

- As accurate as possible so that meaningful engineering design decisions can be made from the tool,
- Extensible so the tool can be expanded to encompass later-developed waveforms and processors, and
- Reusable so development efforts applied to one analysis can be readily applied to another with minimal impact to the user.

Building upon the formal methodology presented in [1], the Processor Cycle Estimation Tool (PCET) provides a reasonably accurate, extensible, and reusable open-source solution to the problems faced in an SDR systems analysis. This paper briefly describes the estimation methods used in PCET in Section 2, the software architecture in Section 3, and the results of some experiments where we compared the values predicted by PCET against those reported by vendors and those we measured ourselves in Section 4.

## 2. PCET METHODOLOGIES

### 2.1 Methodology Overview

PCET builds upon work performed in 2005 to estimate cycles and execution time for specific components on specific processors. The methodology documented in [1] can be visualized as shown in Figure 1 where two different paths are taken to estimate cycles. In one path (in the middle), existing profiled code libraries are used to estimate cycles. While using previously profiled measurements of vendor-supplied code will give the most accurate estimates, the applicability of this path is frequently limited.

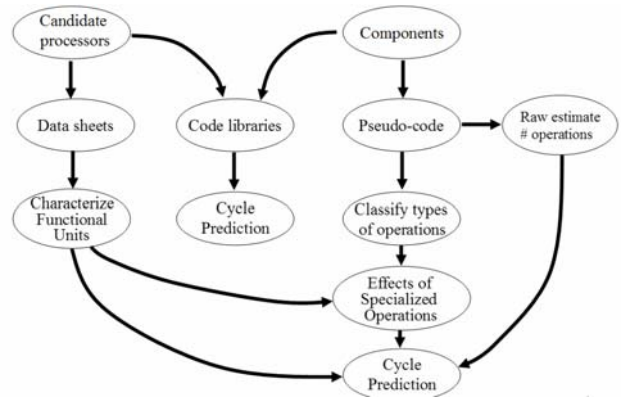


Figure 1: Cycle Estimation Methodology of [1]

The other path attempts to create parameterized implementation equations that describe the resources used when a component is implemented on a specific processor. This begins with an initial operations estimate made by describing the waveform component in pseudo-code intending to capture all relevant operations, including memory accesses, and loop control operations. The total number of operations indicated by the pseudo-code are then tabulated and parameterized (e.g., in terms of block size, radix, or constraint length) to give a raw estimate of the total operations required to support the waveform component. Loosely, these parameterizations are the lengths of the various loops in the code.

Operations related to loop control and other specialized subprocesses are then noted. For a processor that can implement only a single operation at a time, the cycle estimate for a waveform component would simply be the operations estimate. However, most processors include extra circuitry for simultaneously implementing multiple operations. To estimate the cycles a processor uses to implement a particular waveform component, the method subtracts the operations performed in specialized circuitry indicated by the processor's architecture from the operation estimate and adjusts the estimate for architectures that support single-instruction-multiple-data (SIMD) and superscalar instructions, e.g., Very-Long-Instruction-Word (VLIW).

PCET automates much of this process for cycle estimation and then derives the remaining parameters as follows.

- Execution time is calculated as estimated cycles divided by the DSP clock rate.
- Data memory is calculated by counting the number of calls to data memory prior to applying modifiers.
- Program memory is calculated by repeating the cycle calculation with all loop lengths set to 1 without considering the effect of SIMD or VLIW.
- Energy consumption is calculated as Peak DSP power consumption * Execution Time. Note that this is a pessimistic estimate but would be in line with an assumption of hand-coded assembly attempting to minimize processing time (which is therefore leveraging as many resources as possible).

While it is expected that most of the estimations performed by PCET will utilize the second path, by including options for using measured code, PCET can leverage cycle-accurate estimates when available thereby enhancing the accuracy of estimations.

### 2.2 DSP and Component Characterization

To automatically generate parameterized equations that describe the cycles a component will consume on a DSP, both the DSP and the component must be characterized. To characterize a DSP, the DSP's datasheet and instruction set should be studied to identify the following:

1. Peak clock rate
2. Peak power consumption
3. Native bit-field width
4. (For VLIW calculations) Total number of unique instructions that can be completed in one cycle
5. (For VLIW calculations) Total number of memory, arithmetic, and multiplication instructions that can be completed in one cycle (note that the sum of these will sometimes exceed the value found in 4.)
6. (For SIMD calculations) Maximum number of data words to which an instruction can be applied.
7. Any specialized instructions for which two or more "basic" operations are completed in a single cycle (e.g., a MAC or zero-overhead).
8. Addressing modes, number representation formats

Characterizing a component requires a more creative process. First, the component should be expressed in a pseudo-assembly style that captures every operation required to implement a component on the least capable DSP known at the time. Second, any DSP characteristics assumed to exist when writing the pseudo-code such as minimum bit-field width (for fixed point components), addressing modes (e.g., circular or bit-reversed), or other number formats (e.g., rounding modes) should be documented. Third, any parameterizable aspects of the pseudo-code length (typically loop lengths) should be noted.

Fourth, each line should be numbered, with a unique numbering style applied to each loop. Using this numbering, expressions should be written that express the number of data memory operations, the number of multiplications, and the number of arithmetic operations. Then the pseudo-code is reviewed to identify if / where previously characterized specialized operations apply (see DSP characterization step # 6). This allows us to define an expression for the number of operations that should be subtracted by PCET from the total when a DSP with that specialized operator is targeted. Because some specialized operations will target the same line, these "synergistic" effects need to be undone. So by reviewing the line numbers where these duplicated eliminations occur, additional expressions to add back in the doubly (or triply) eliminated operations are written. To aid the VLIW estimation routine in PCET, the type of operation (e.g., data memory multiplications, or arithmetic) in each equation should be noted. The VLIW and SIMD methods used by PCET are documented in [2], but their implementation is not important to the user's DSP and component characterization process. Finally, if the analyst knows of any previously profiled code of the component on a particular DSP, those profiled expressions for cycles, program memory and data memory should be included.

### 2.3 Example Characterization

The following gives an example of a characterization of a FIR filter of arbitrary length which calculates outputs on a sample-by-sample basis where the samples and coefficients are real-valued. The component characterization begins by writing the pseudo-code shown in Figure 2. Note that this implementation assumes the existence of a circular addressing mode and that the parameterizable aspect of this code is the filter length, which we will assign the variable name 'N'. Also note that to support the widest array of DSP's possible, the simplest loop control operation was assumed in the pseudo-code wherein three instructions are used: one to decrement the loop counter, one to compare the loop counter with zero, and a conditional branch based on the result of that comparison.

```
y=fir(coef, data, length, offset)

//Set circular buffer params
1     (instruction to store previous setting in local register)
2     (instruction to store buff length)
3     (instruction to turn on circ buff)
4     (instruction to set buffer length)

//Move input parameters to local registers
5     R1 = coef (address)
6     R2 = data (address)
7     R2 = data + offset // needed for circular buffering
8     R3 = length (actual #)

//zero accumulator (typically done by subtracting a register from itself)
9     acc = 0

//Note inherent assumption that length > 0
//Note for loops are implemented as conditional branches in assembly
L1    (loop label)      R4 = *R1++ //postfix assumption
L2    R5 = *R2++ (
L3    R6 = R5 * R4
L4    acc = acc + R6
L5    R3 = R3 – 1
L6    flag = cmp(R3,0)
L7    if flag (R3==0), branch to loop

//    Move result to output register
10    R_out = acc

//    Restore stuff
11    (instruction to turn reset addressing mode)
12    (instruction to reset buffer length)
13    (instruction to branch back)
```

Figure 2: Pseudo-code for a process that implements a sample-by-sample filter with an arbitrary number of real-valued coefficients

Next, the pseudo-code is numbered, where looped lines are labeled as 'L#'. Each of the operations were then classified and combined to form the four equations shown in Table 1 where the total equation is strictly not necessary, but is useful for bookkeeping.

Table 1: Raw Operations Equation

| Class | Equation |
|---|---|
| Total | 13+ 7 * N |
| Memory | 2*N |
| Multiplication | N |
| ALU | 4*N + 13 |

Then the known specialized instructions are reviewed, and the lines where they apply are noted as well as what kind of operation is being targeted. These are summarized in Table 2 where BDEC is branch and decrement, BPOS is a branch if positive, ZOL is zero-overhead loop (called block repeat on other processors), MEM2 represents double wide memory loads, and NOREG is used to model C54 behavior wherein local registers are eschewed in favor of direct data memory accesses (this has the effect of slowing the peak clock rate achievable by the processor).

Table 2: Impact of Specialized Instructions

| Instruction | Impact | Modifier Equation |
|---|---|---|
| BDEC | L5, L6 eliminated | (ALU) -2*N |
| BPOS | L6 eliminated | (ALU) – N |
| ZOL | 1 cycle to set register, L5, L6, L7 eliminated | (ALU) 1 – 3*N |
| MAC | L4 eliminated | (ALU) -N |
| MEM2 | MEM cut in half | (MEM) –(2*N)/2 |
| MEM4 | MEM cut in fourth | (MEM) -3*(2*N)/4 |
| NOREG | MEM eliminated | (MEM) –(2*N) |

As several of these modifiers target the same line, these synergistic effects have to be undone so that a line is only removed from consideration once. Equations relating these are shown in Table 3.

Table 3: Synergistic Modifiers for Real FIR Mapping

| Modifiers | Impact | Modifier Equation |
|---|---|---|
| BDEC, ZOL | L5,L6 added back in | (ALU) target+2*N |
| BPOS, BDEC | L6 added back in | (ALU) target+N |
| BPOS, ZOL | L6 added back in | (ALU) target+N |
| BPOS, BDEC, ZOL | L6 eliminated (again) | (ALU) target – N |
| MEM2, NOREG | MEM2 effect undone | (MEM) target + (2*N)/2 |
| MEM4, NOREG | MEM4 effect undone | (MEM) target + 3*(2*N)/4 |

## 3. TOOL DESCRIPTION

### 3.1 Software Overview

As conceptually illustrated in Figure 3, PCET is implemented as four primary components.

1. A collection of DSP and component characterization files.
2. A GUI to manage the component and DSP characterization files and to generate parameterization files. The parameterization files are used to specify particular values to substitute into the component characterization equations, e.g., the filter length in the example in Section 2. The GUI also provides a mechanism for saving and loading results and analysis configurations.
3. The computational engine which generates estimates of cycle, time, energy, and memory values for parameterized combinations of DSPs and components
4. A results window for tabulating the results and flagging any mappings that exceed specified limits (e.g., taking too long to execute).

### 3.2 Key Tool Features

The GUI is designed such that any DSP or component characterization files placed in the proper directories are automatically detected at initialization and included in the set of available characterizations. Additionally, the engine maintains no internal characterization information beyond the input files given to it. These features along with light characterization file error trapping enhance the extensibility of PCET and allow new characterization files to be written without knowledge of the underlying code structure.

While a highly extensible solution, the initial PCET release also includes a significant number of characterizations. This includes 20 different families of processors drawn from TI, Analog Devices, ARM, Intel, PowerPC, and Freescale and 24 components which implement filters, multi-rate processes, synchronization, error correction, transcendental number generation, FFTs, digital modulation and demodulation, and analog modulation and demodulation.

To help designers consider tradeoffs, multiple copies of the same component can be selected and assigned different parameters to allow for sensitivity analysis (e.g., impact of traceback length on run-time).

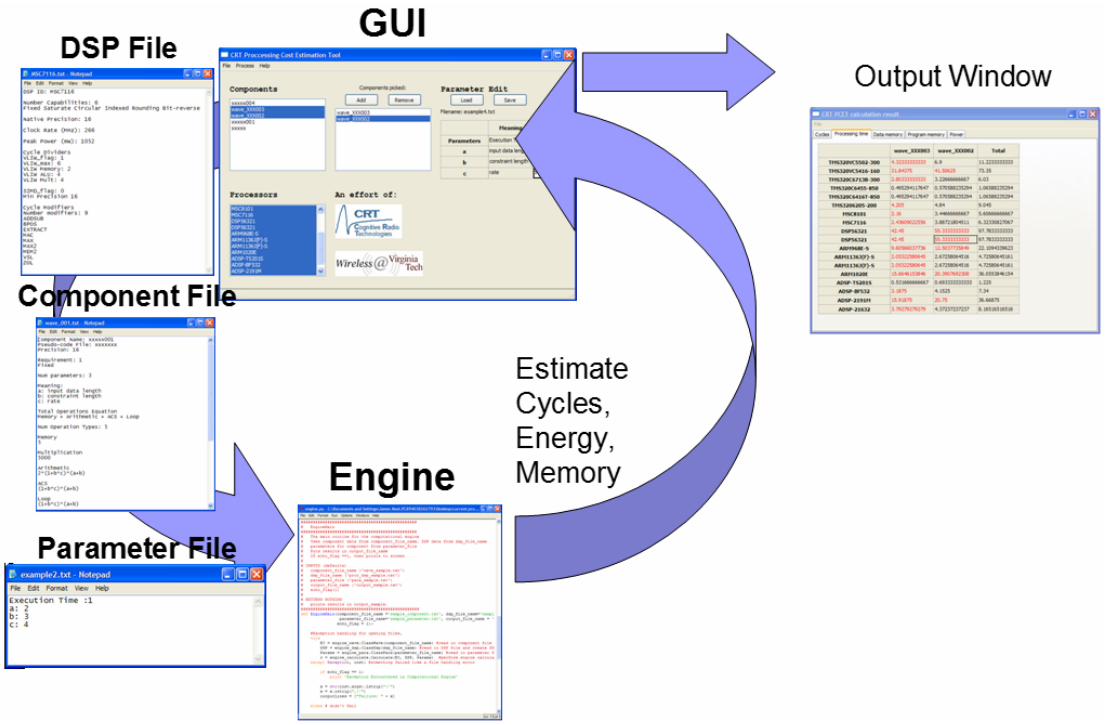


Figure 3: Conceptual operation of PCET

## 4. VALIDATION

In an effort to validate the estimations of PCET, we compared estimated values against vendor-reported measurements and measurements performed at Virginia Tech.

### 5.1. Library Validation

We reviewed vendor provided code libraries for components whose assumptions most closely matched the assumptions used in creating the component characterization files. We judged the assumptions used in the real filter (sample-by-sample, arbitrary filter lengths), the complex filter (sample-by-sample, arbitrary filter lengths), the complex FFT (radix-2 using precomputed twiddle-factors), and the real LMS (symbol-by-symbol filter with weight update, but no error calculation) of the library code used by the TMS320VC5502 [3], TMS320VC5416 [4], and the TMS3206205 [5] to be reasonable (though necessarily imperfect) matches to the assumptions used in the PCET component characterizations. The following reports the results of the comparisons of PCET estimations against the equations supplied with vendor library code with the additional caveat that library cycles needed for more generalized overhead (e.g., stack management) was ignored in the comparison as it was not modeled in the component characterizations.

As shown in Table 4, real filters were varied for filter lengths 15, 31, 63, and 127. The cycles required by the C54 (fir) and C55 (fir2) grew at exactly the same rate which implies that the code kernels agree perfectly. The cycles required by the C62 (fir_gen) grew at approximately twice the rate of the PCET estimation which implies that the library code is not making full use of the dual multipliers.

As shown in Table 5, complex filters were varied for filter lengths 15, 31, 63, and 127. Only the C54 (cfir) had library code available. The C54 library code required approximately twice as many cycles as the PCET estimate.

As shown in Table 6 Real LMS Filters were varied for lengths 7, 15, and 31. Only the C54 (dlms) had library code available. The C54 library code required only differed by one cycle from the PCET estimate.

As shown in Table 7, complex FFTs were varied for block lengths 32, 64, 128, and 256. Only the C54 (cfft) and C55 (cfft) had library code available. The C54 library code required a similar number of cycles as estimated by PCET (remarkable given the complexity) while the C55 differed significantly. Oddly, the C55 library code required significantly more cycles than the C54 code, which is surprising as the C55 is the more sophisticated processor. Likely, there are hidden switches in the library code which are adding significantly to the code estimates. Note that the C62 did not support the bit-reverse addressing assumed in the component characterization, so no estimates were made.

Table 4: Real Filter Cycles Data

| DSP | Method | 15 | 31 | 63 | 127 |
|---|---|---|---|---|---|
| VC5502 fir2 | Library | 11 | 19 | 35 | 67 |
| | PCET | 21 | 29 | 45 | 77 |
| VC5416-160 fir | Library | 23 | 39 | 71 | 135 |
| | PCET | 29 | 45 | 77 | 141 |
| 6205-200 fir_gen | Library | 40 | 72 | 136 | 264 |
| | PCET | 12 | 24 | 48 | 96 |

Table 5: Complex Filter Cycles Data

| DSP | Method | 15 | 31 | 63 | 127 |
|---|---|---|---|---|---|
| VC5502 N/A | Library | N/A | N/A | N/A | N/A |
| | PCET | 55 | 87 | 151 | 279 |
| VC5416-160 cfir | Library | 133 | 261 | 517 | 1029 |
| | PCET | 85 | 149 | 277 | 533 |
| 6205-200 N/A | Library | N/A | N/A | N/A | N/A |
| | PCET | 30 | 59 | 118 | 235 |

Table 6: Real LMS Cycles Data

| DSP | Method | 7 | 15 | 31 |
|---|---|---|---|---|
| VC5502 N/A | Library | N/A | N/A | N/A |
| | PCET | 21 | 29 | 45 |
| VC5416-160 dlms | Library | 28 | 44 | 76 |
| | PCET | 29 | 45 | 77 |
| 6205-200 N/A | Library | N/A | N/A | N/A |
| | PCET | 13 | 26 | 53 |

Table 7: FFT Cycles Data

| DSP | Method | 32 | 64 | 256 | 1024 |
|---|---|---|---|---|---|
| VC5502 cfft | Library | 517 | 1036 | 4858 | 23848 |
| | PCET | 406 | 730 | 2594 | 9930 |
| VC5416-160 cfft | Library | 390 | 806 | 3302 | 13286 |
| | PCET | 431 | 773 | 2737 | 10461 |
| 6205-200 N/A | Library | N/A | N/A | N/A | N/A |
| | PCET | N/A | N/A | N/A | N/A |

### 5.2 An Independent Validation Study

Researchers at Virginia Tech recently compared the estimates provided by PCET against profiled measurements from the PowerPC 405D5. As the PowerPC 405D5 was not one of the initially characterized DSPs, this necessitated writing a new DSP characterization file. Efforts were made to translate the documented module assembly for selected component characterizations [2] into C which was then compiled and executed on the PowerPC 405D5. To measure the length of run time on the 405D5, the processor's 64-bit timer register is accessed before and after running each component 500 times and storing the difference. This allowed for finer grained measurements of the estimated execution time and measurements for components that require less than 1 us to execute. This was performed for 5 components (real filter, complex filter, LMSReal, Taylor, and CORDIC) and the results are shown below. Because this is compiled C code, it would be expected that PCET (which is modeling aggressive hand-coded assembly) would

provide smaller estimates. Largely, this is what happened, with significant variance between surveyed processes.

Of more immediate relevance, this exercise demonstrated the ease of third-party development of characterization files for PCET as this effort was performed entirely independently of the original PCET developers in two weeks with approximately 2 days for characterization of an unfamiliar architecture. Note that once the PowerPC characterization file was complete, the implementation statistics for all 24 component files could be immediately estimated.

Table 8: PCET estimates and profiled measurements from compiled C code for selected components on the PowerPC 405D5

| Component | Length | PCET Cycles | PCET Time | Profiled Time |
|---|---|---|---|---|
| RealFilter | 20 | N/A | N/A | 0.41 us |
| ComplexFilter | 10 | N/A | N/A | 0.41 us |
| LMSReal | 10 | 163 | 0.54 us | 8.28 us |
| Taylor | 10 | 86 | 0.29 us | 0.31 us |
| CORDIC | 14 | 289 | 0.96 us | 0.21 us |

## 6. CONCLUSIONS

PCET is an open-source tool available for download from CRT [2] as an independent software package under a Mozilla Public License and from Virginia Tech as an OSSIE tool. Thus other users can extend PCET to new applications, fix bugs, or simply add new component and DSP characterizations. Particularly, as more characterizations are created, the PCET approach of automatically estimating cycles, execution time, energy, and memory will become increasingly valuable as each new characterization can leverage existing characterizations. This effect was demonstrated when Virginia Tech coded a new DSP characterization that could immediately leverage all 24 pervious component characterizations. In this way, we believe PCET will radically improve SDR systems engineering by allowing for more rapid analysis of porting and implementation feasibility and of parameter sensitivities.

While intended as a design tool for human radio designers, PCET could also act as a key internal model for cognitive radios to estimate the impact of various design decisions on internal radio resources. The rapidity by which PCET makes its estimations of power, cycles, memory, and implementation feasibility (on the order of microseconds on a typical GPP) implies that the addition of this process would have little impact on radio operation while providing a significant new feature.

In general, PCET performs quite well when the assumptions of a characterized component closely match the assumptions of the implemented component. Though not revealed in these validation studies, it is believed that some implicit assumptions in the PCET cycle estimation methodology will lead to underestimates of required cycles for RISC processors where instruction executions times depend on what other instructions are vying for processor resources. In contrast, it is also believed that as the number of parallelizable options increase on a processor, the predefined VLIW and SIMD algorithms will overestimate the required number of cycles.

Beyond the initial PCET release, several immediate avenues for improvement are available including:

- automating the generation of component characterization files from a high level language description or generating DSP characterization files from datasheets
- augmenting the GUI to support a more SDR-systems-engineer-friendly block diagram layout that allows definition of timing domains and possibly processor boundaries
- Support for FPGAs
- Explicit support for multi-core DSPs.
- Integration into other packages such as Simulink or another simulation package so that both resource and performance statistics could be gathered in one package
- A RISC-specific estimation model to address expected shortcomings in the engine.

## 7. ACKNOWELDGMENTS

This work was sponsored by CERDEC through Naval Surface Warfare contract N00178-98-D-3017-0044.

## 8. REFERENCES

[1] J. Neel, P. Robert, J. Reed, “A Formal Methodology for Estimating the Feasible Processor Solution Space for a Software Radio,” SDR Forum Technical Conference, 2005.

[2] PCET Resource Webpage: www.crtwireless.com/PCET.html

[3] TMS320C55x DSP Library Programmer’s Reference, SPRU422I, August 2006.

[4] TMS320C54x DSP Library Programmer’s Reference, SPRU518d October 2004.

[5] TMS320C62x DSP Library Programmer’s Reference, SPRU402B, October 2003.