Non-Intrusive Runtime Monitoring Through Power Consumption: A Signals and System Analysis Approach to Reconstruct the Trace (Appendices not included in the conference proceedings)

Appendix A Instrumentation of the Source Code – See §3.4

Below are examples of the two instrumented versions of the source code for the case of the ADPCM coder.

Print-instrumented version:

```
void adpcm_coder(short indata[],
                 char outdata[], int len,
                 struct adpcm_state * state)
{
    short *inp;
        /* Input buffer pointer */
    signed char *outp;
        /* output buffer pointer */
    /* ... other declarations */

printf ("Node0x20ccb50\n");
    outp = (signed char *)outdata;
    inp = indata;
    valpred = state->valprev;
    index = state->index;
    step = stepsizeTable[index];
    bufferstep = 1;

    for ( ; len > 0 ; len-- )
    {
printf ("Node0x20ccea0\n");
        val = *inp++;
        diff = val - valpred;
        /* ... */
    }
    /* ... */
}
```

Flip-port-bit-instrumented version:

```
extern char volatile port_bit;
#define FLIP_PORT_BIT \
  {PORTG = (port_bit = !port_bit);}

void adpcm_coder(short indata[],
                 char outdata[], int len,
                 struct adpcm_state * state)
{
    short *inp;
        /* Input buffer pointer */
    signed char *outp;
        /* output buffer pointer */
    /* ... other declarations */

FLIP_PORT_BIT;
    outp = (signed char *)outdata;
    inp = indata;
    valpred = state->valprev;
    index = state->index;
    step = stepsizeTable[index];
    bufferstep = 1;

    for ( ; len > 0 ; len-- )
    {
FLIP_PORT_BIT;
        val = *inp++;
        diff = val - valpred;
        /* ... */
    }
    /* ... */
}
```

Appendix B Randomized Sequences of Functions – See §4.1

Below is an example of a randomized sequence of functions. The program running on Workstation 1 randomly chooses the 64-bit seed for the `rnd64` PRNG, as well as the choice of functions at each step (for example, `encrypt` and `crc32buf` were randomly chosen for the first step, `sha_update` and `adpcm_coder` for the second step, and so on).

The function `randomize_data` uses `rnd64` to generate pseudorandom input data for the functions. Every eight steps (eight `if` statements) we re-randomize and assign a new random value into `rnd`, since each step consumes one of its eight random bits.

```
srnd64(UINT64_C(8973546545337244988));
uint8_t rnd;

randomize_data();
rnd = ((rnd64() >> 24) & 0xFF);
if (rnd & 0x1)
    encrypt (plaintext, ciphertext, &ctx);
else
    rc = crc32buf (crcdata, CRCSIZE);
rnd >>= 1;


if (rnd & 0x1)
    sha_update (&sha_info, sha_data, SHASIZE);
else
    adpcm_coder (pcmdata, adpcmdata, PCMSIZE,
                 &coder_1_state);
rnd >>= 1;

if (rnd & 0x1)
    fft_float (FFTSIZE, 0, real_in, imag_in,
                           real_out, imag_out);
else
    sha_update (&sha_info, sha_data, SHASIZE);
rnd >>= 1;

...
```