ASIP Lab

Kostas Dagkas
Sebastian Gregorzyk

I Performance

Algorithm - adpcm.c

```
/* Step 4 - Compute difference and new predicted value */
/*
** Computes 'vpdiff = (delta+0.5)*step/4', but see comment
** in adpcm_coder.
*/

vpdiff = step >> 3;
if ( delta & 4 ) vpdiff += step;
if ( delta & 2 ) vpdiff += step >> 1;
if ( delta & 1 ) vpdiff += step >> 2;
```

Algorithm - adpcm.c

```
static int stepTable[89][8] =
{
    {0x0000, 0x0001, 0x0003, 0x0004, 0x0007, 0x0008, 0x000a, 0x000b},
    {0x0001, 0x0003, 0x0005, 0x0007, 0x0009, 0x000b, 0x000d, 0x000f},
    {0x0001, 0x0003, 0x0005, 0x0007, 0x000a, 0x000c, 0x000e, 0x0010},
    >>>                          ...                              <<<
    {0x0d39, 0x27ac, 0x4220, 0x5c93, 0x7707, 0x917a, 0xabee, 0xc661},
    {0x0e8c, 0x2ba4, 0x48bd, 0x65d5, 0x82ee, 0xa006, 0xbd1f, 0xda37},
    {0x0fff, 0x2ffe, 0x4ffe, 0x6ffd, 0x8ffe, 0xaffd, 0xcffd, 0xeffc}
};

vpdiff = stepTable[index][(delta & 7)];
```

One New Instruction

```
decode[R|L] rd, rs0, rs1
```

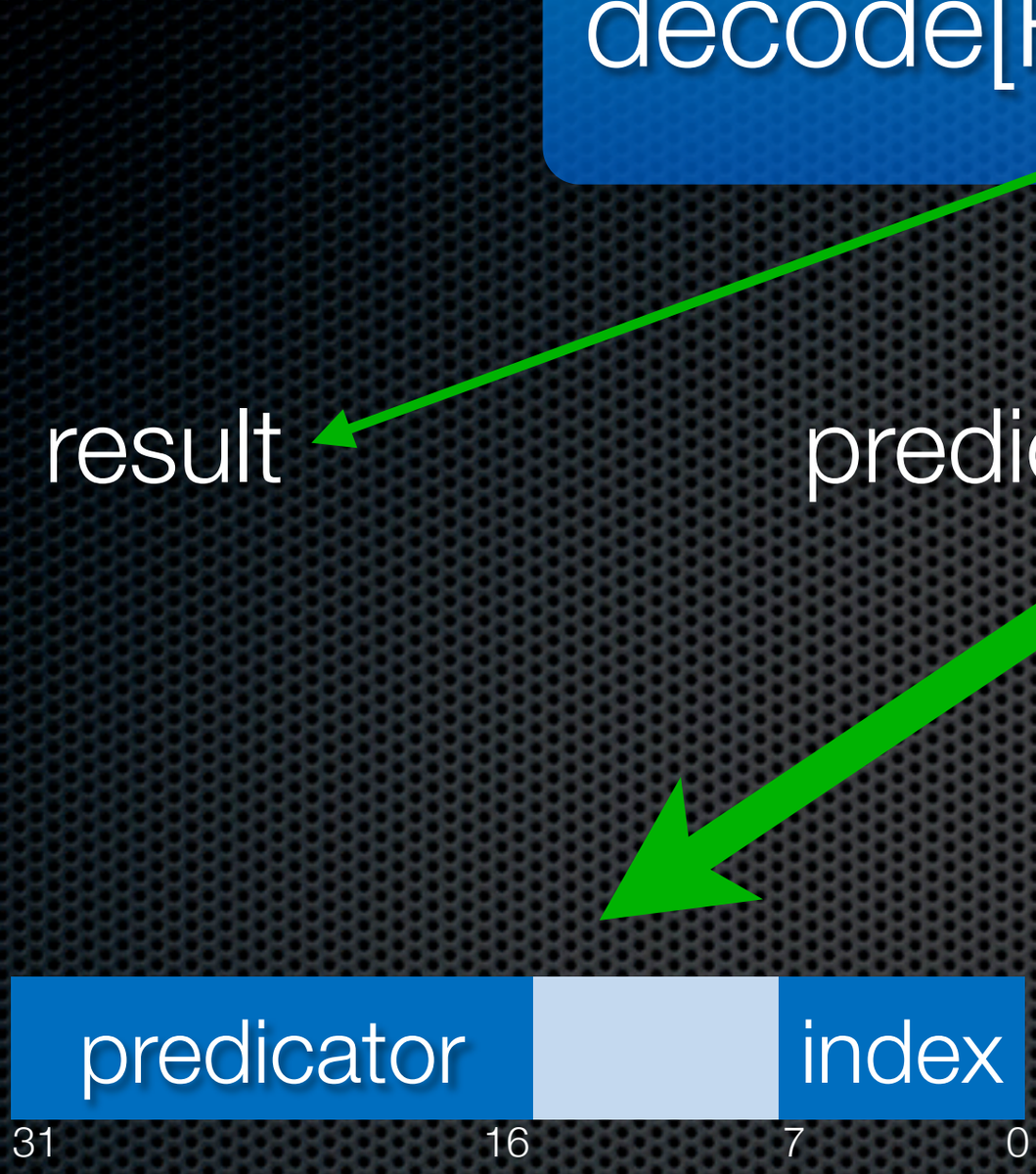
- decodes one audio sample (left or right channel)
- writes result to memory mapped DAC

decode[R|L] rd, rs0, rs1

result

predicator & index

stereo sample



3 New Hardware Resources

- INDX LUT + saturating adder
- STLUT lookup table
- SATADD saturating adder

INDX resource


```
static int indexTable[16] =
{
    -1, -1, -1, -1, 2, 4, 6, 8,
    -1, -1, -1, -1, 2, 4, 6, 8,
};

index += indexTable[delta];
if(index < 0) index = 0;
if(index > 88) index = 88;
```

STLUT resource


```
static int stepTable[89][8] =
{
    {0x0000, 0x0001, 0x0003, 0x0004, 0x0007, 0x0008, 0x000a, 0x000b},
    {0x0001, 0x0003, 0x0005, 0x0007, 0x0009, 0x000b, 0x000d, 0x000f},
    {0x0001, 0x0003, 0x0005, 0x0007, 0x000a, 0x000c, 0x000e, 0x0010},
    >>>                          ...                          <<<
    {0x0d39, 0x27ac, 0x4220, 0x5c93, 0x7707, 0x917a, 0xabee, 0xc661},
    {0x0e8c, 0x2ba4, 0x48bd, 0x65d5, 0x82ee, 0xa006, 0xbd1f, 0xda37},
    {0x0fff, 0x2ffe, 0x4ffe, 0x6ffd, 0x8ffe, 0xaffd, 0xcffd, 0xeffc}
};

vpdiff = stepTable[index][(delta & 7)];
```

SATADD resource


```
if(sign) valpred -= vpdiff;
else valpred += vpdiff;

if (valpred > 32767) valpred = 32767;
else if (valpred < -32768) valpred = -32768;
```


```
current_pc = PC.read();
inst = IMAU.read(current_pc);
null = IR.write(inst);
null = PC.inc();
```


```
source0 = GPR.read0(rs0);
source1 = GPR.read1(rs1);
predicator = source0[31:16];
index_in = source0[7:0];
deltaR = source1[7:4];
```


```
index_out = INDX0.new(index_in, deltaR);
sign = deltaR[3];
deltaR_tmp3 = deltaR[2:0];
diff = STLUT0.get(index_in, deltaR_tmp3);
result = SATADD0.add(diff, predicator, sign);
```


```
zero16 = "0000000000000000";
result15 = result[14:0];
msb = result[15];
toggle = ~msb;
addr = "0000001001100010010110100000100";
data = <zero16, toggle, result15>;
addr_err = DMAU.store(addr, data);
```


```
zero8 = "00000000";
index16 = <zero8, index_out>;
output = <result, index16>;
null = GPR.write0(rs0, output);
null = GPR.write1(rd, output);
```

adpcm.c

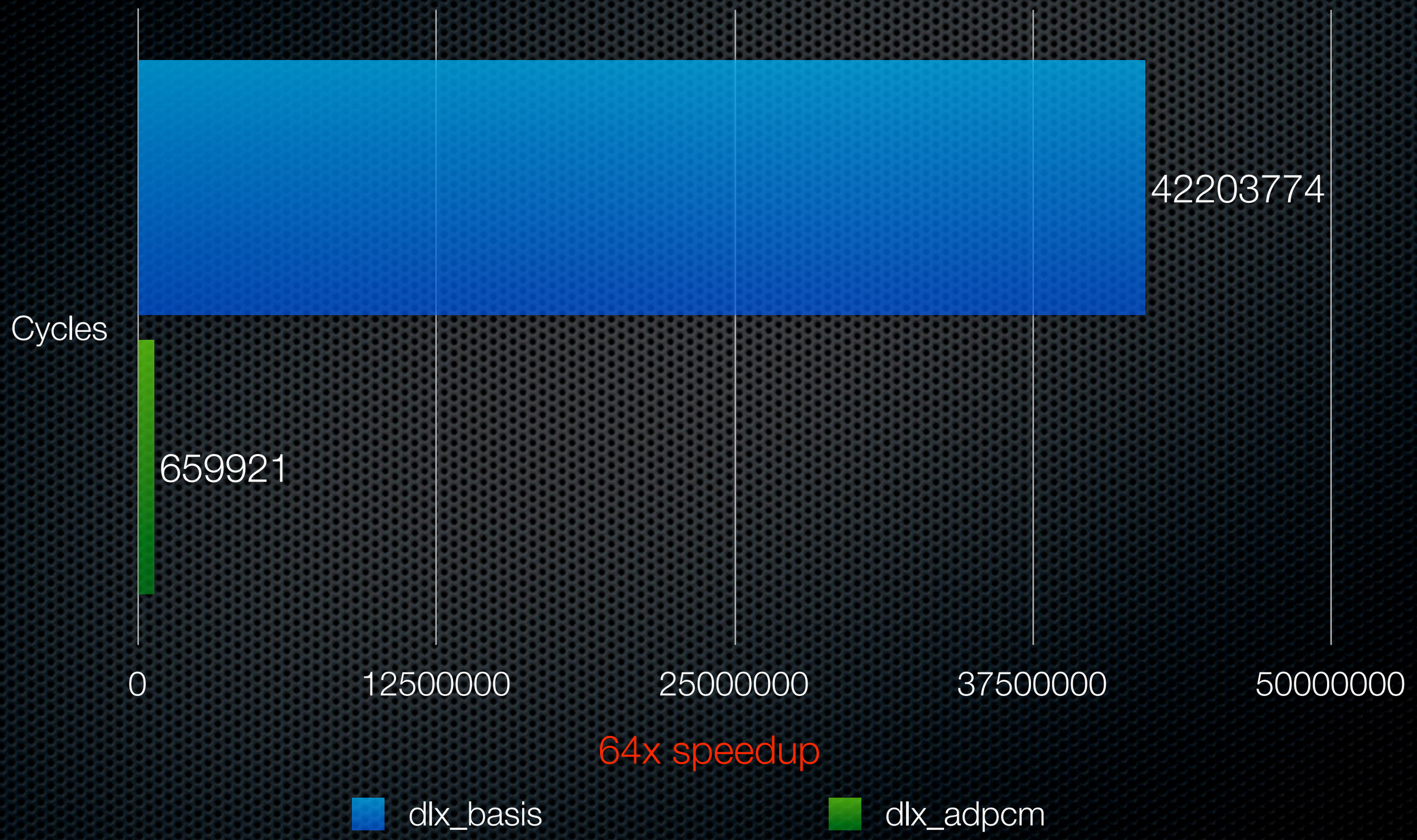
```
int adpcm_decoder(unsigned char* indata, int len)
{
    int inputbuffer = 0;    /* sample data */
    int predidx = 0;        /* predicator & index */

    for ( ; len > 0 ; len-- )
    {
        inputbuffer = *indata++;

        decodeR(predidx, inputbuffer);

        decodeL(predidx, inputbuffer);
    }

    return 0;
}
```

adpcm.s

```
decode_loop:
    decodeR   %GPR3, %GPR3, %GPR2
    sne       %GPR27, %GPR1, %GPR4
    lbu       %GPR2, 0(%GPR1)
    addui     %GPR1, %GPR1, $1
    decodeL   %GPR3, %GPR3, %GPR2
    bnez      %GPR27, decode_loop
```

Benchmarks

Decoding 120.000 samples



64x speedup

dlx_basis

dlx_adpcm

Area

Slices

2721

3274

0

825

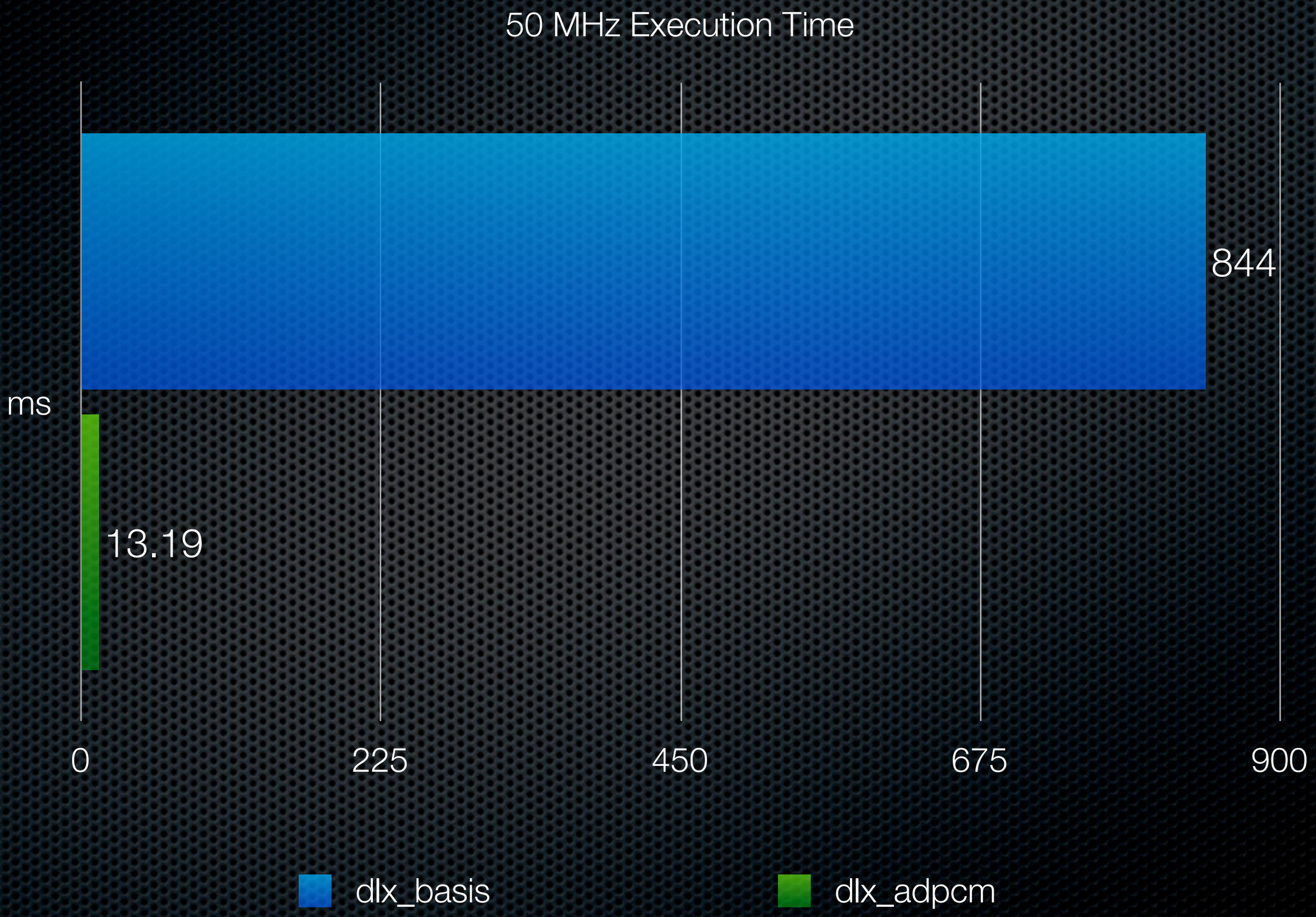
1650

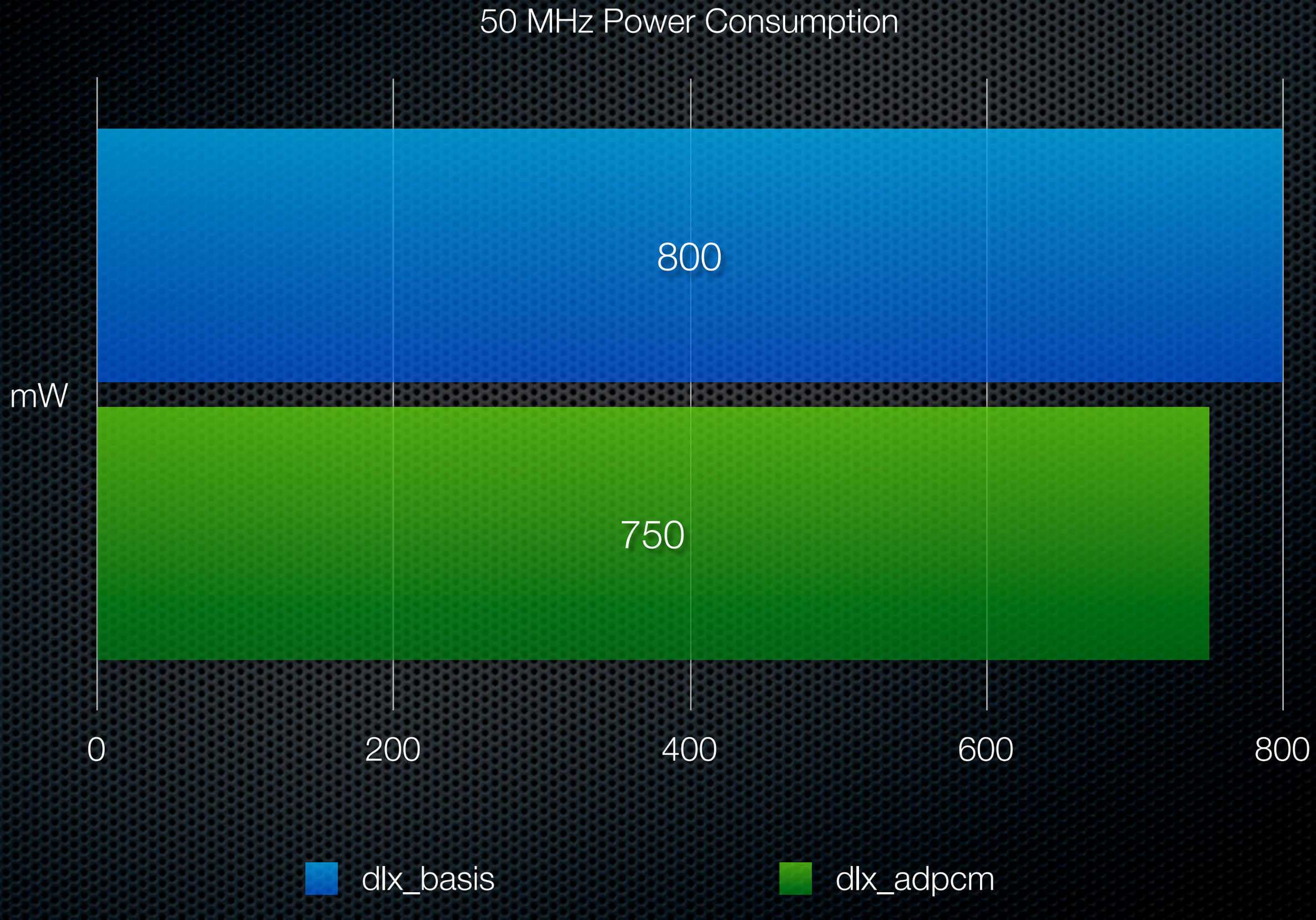
2475

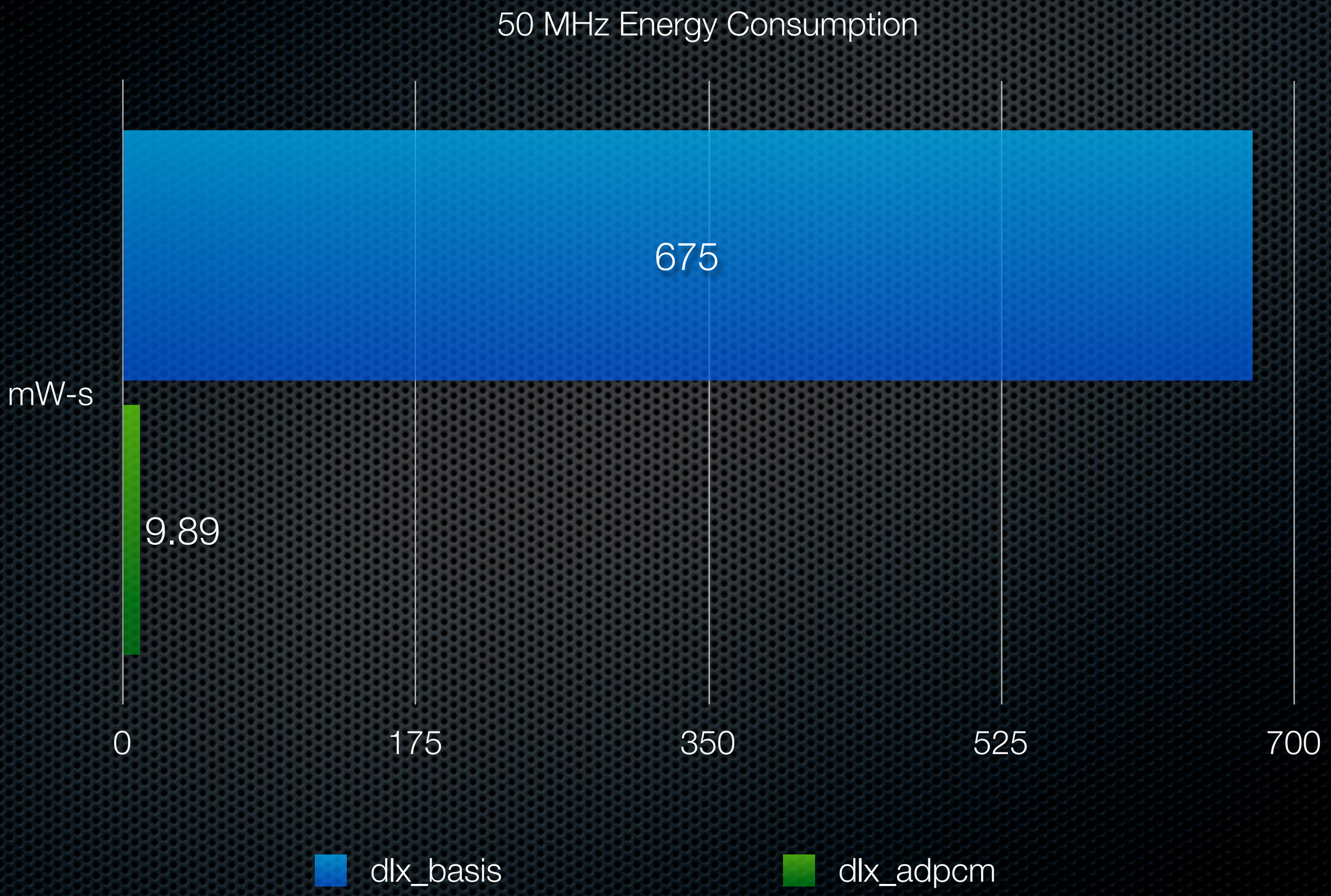
3300

dlx_basis

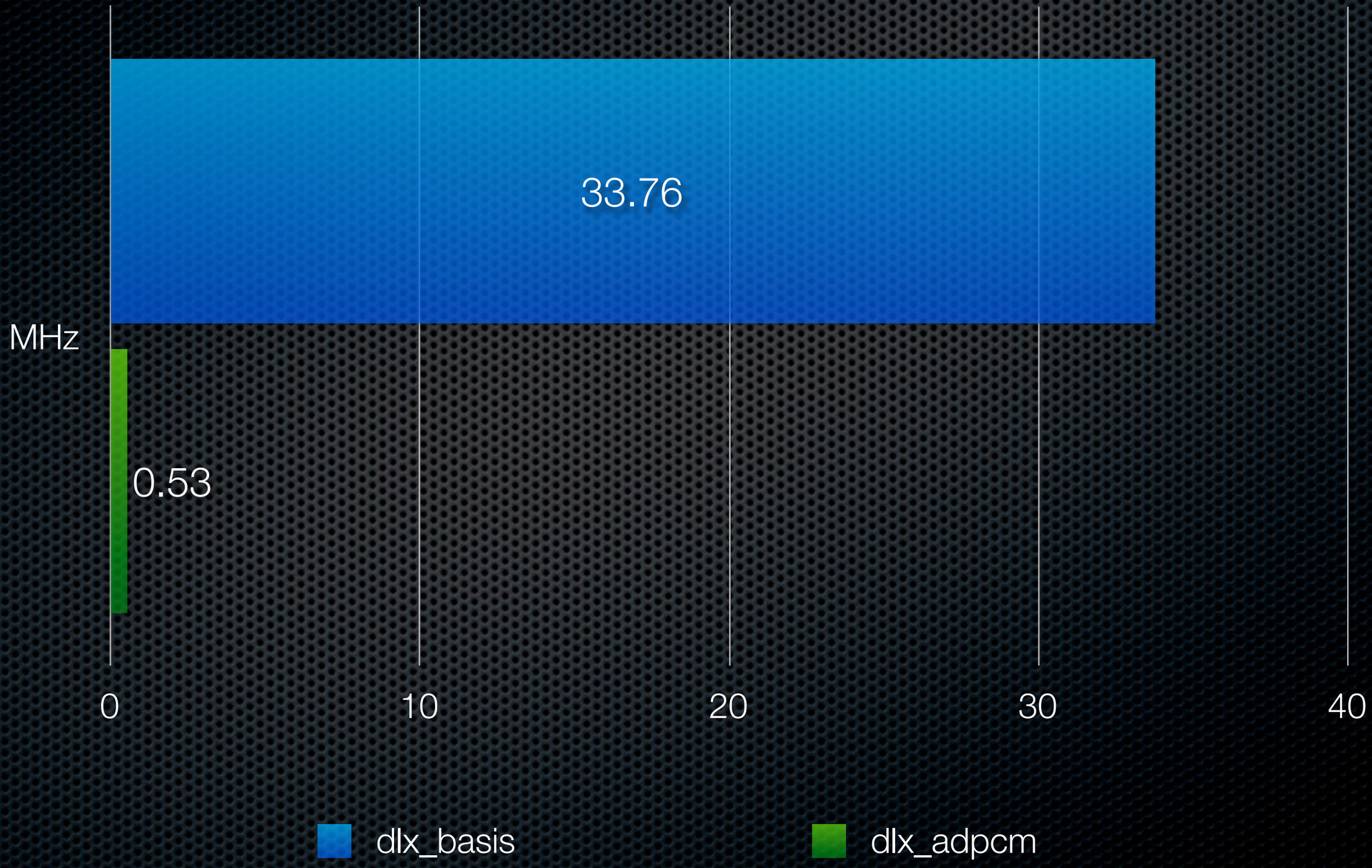
dlx_adpcm


50 MHz Execution Time
844
ms
13.19
0
225
450
675
900
dlx_basis
dlx_adpcm

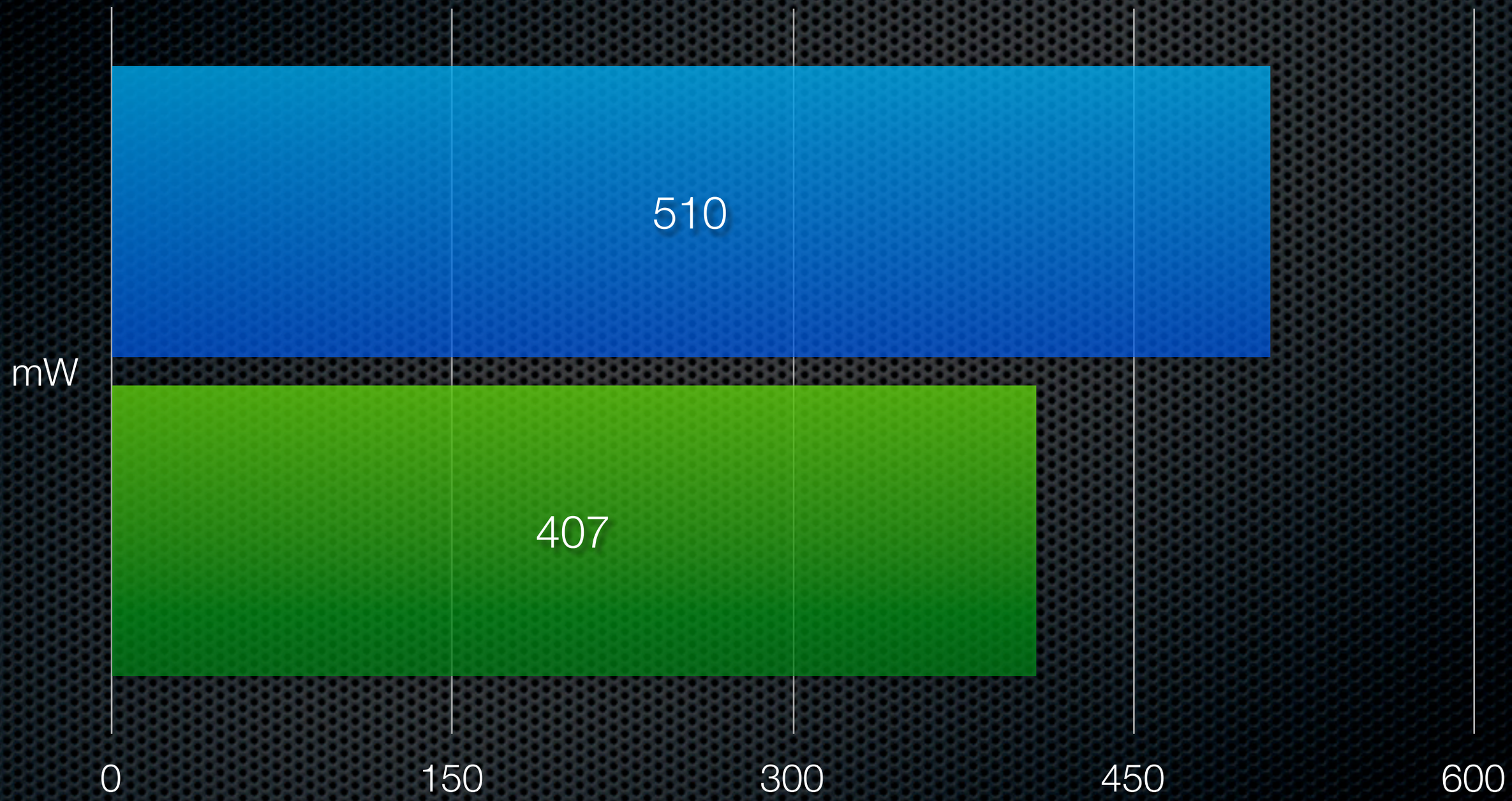

50 MHz Power Consumption
800
750
mW
0
200
400
600
800
dlx_basis
dlx_adpcm


50 MHz Energy Consumption
675
mW-s
9.89
0
175
350
525
700
dlx_basis
dlx_adpcm

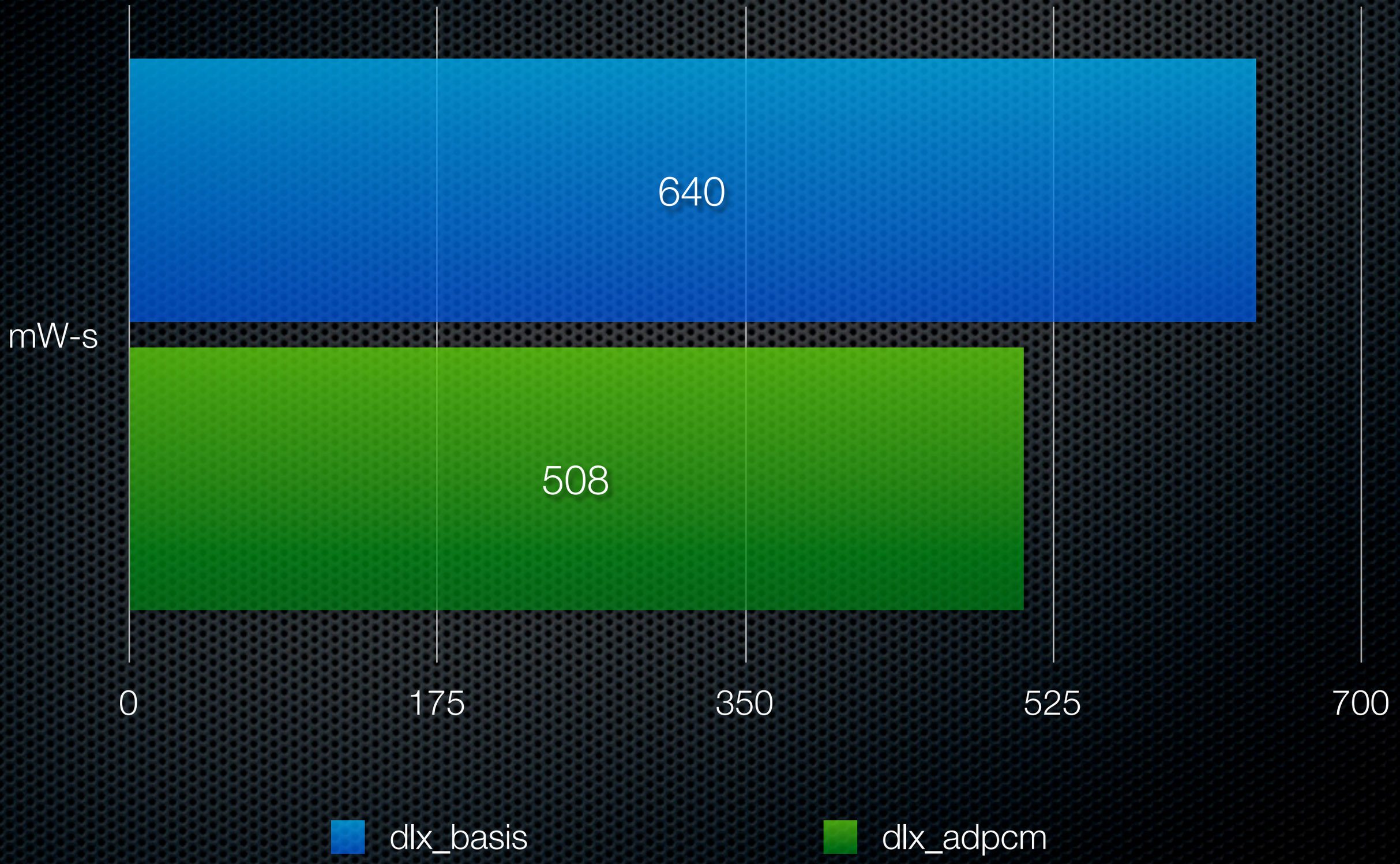
Minimum Frequency



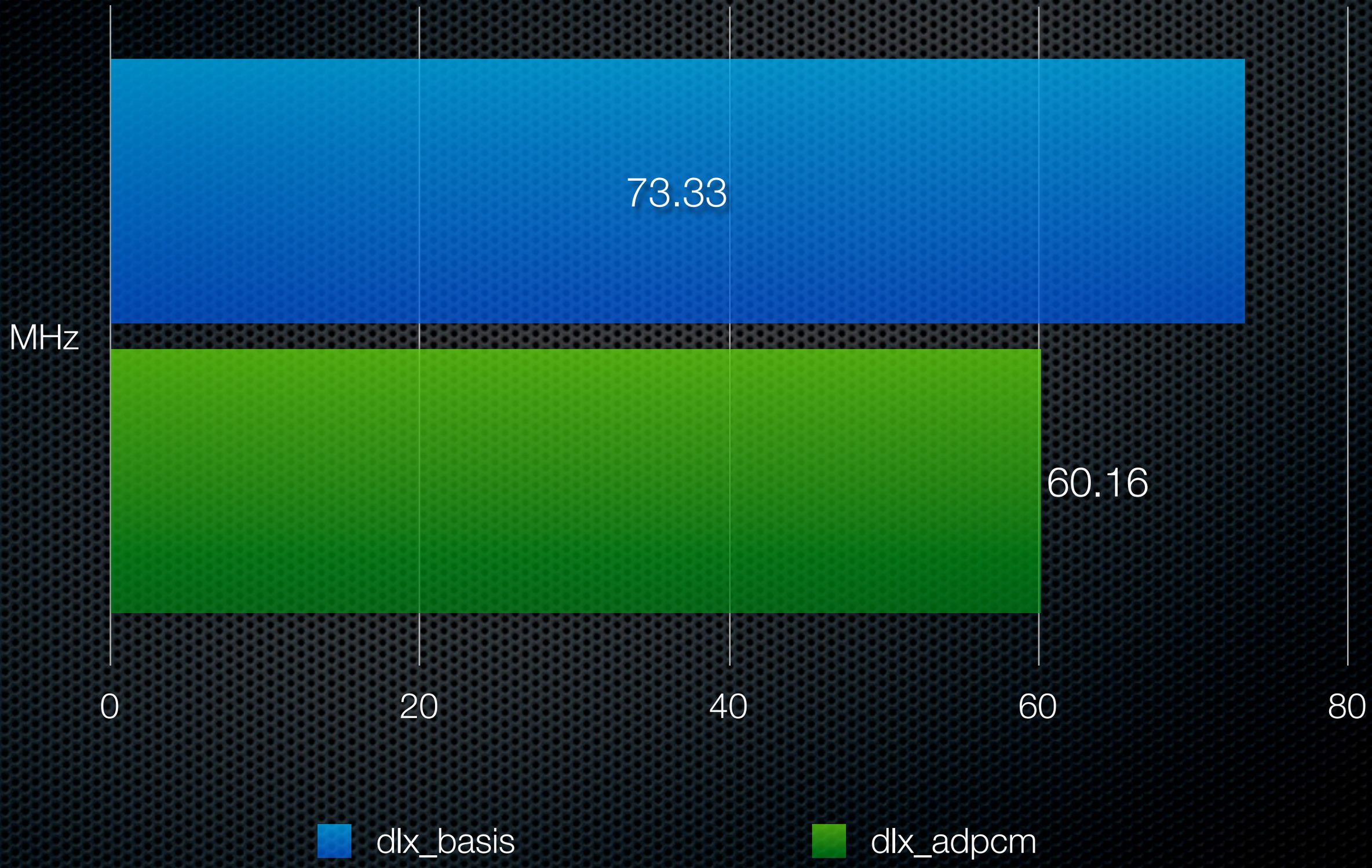
Minimum Frequency Power Consumption



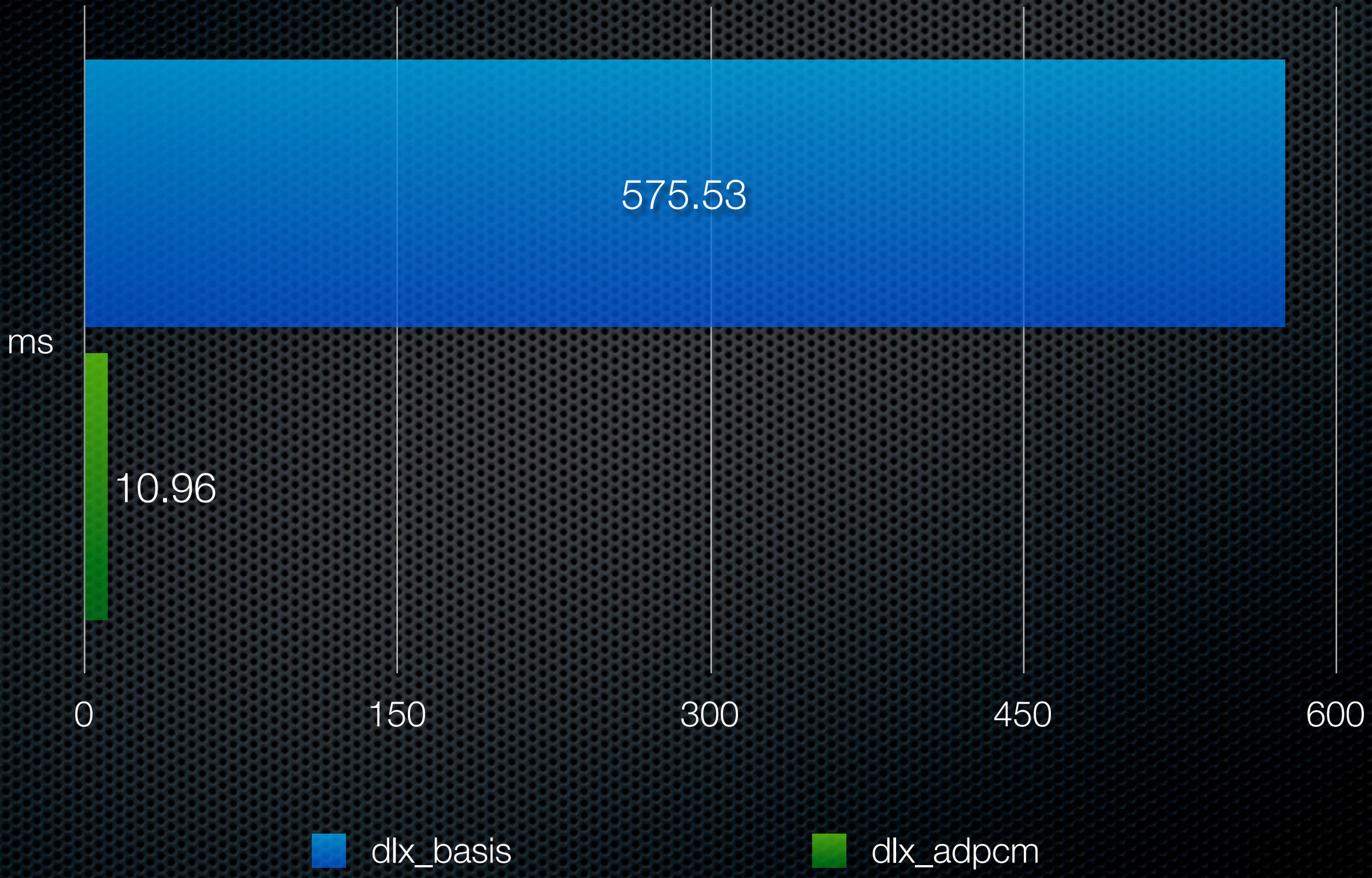
Minimum Frequency Energy Consumption



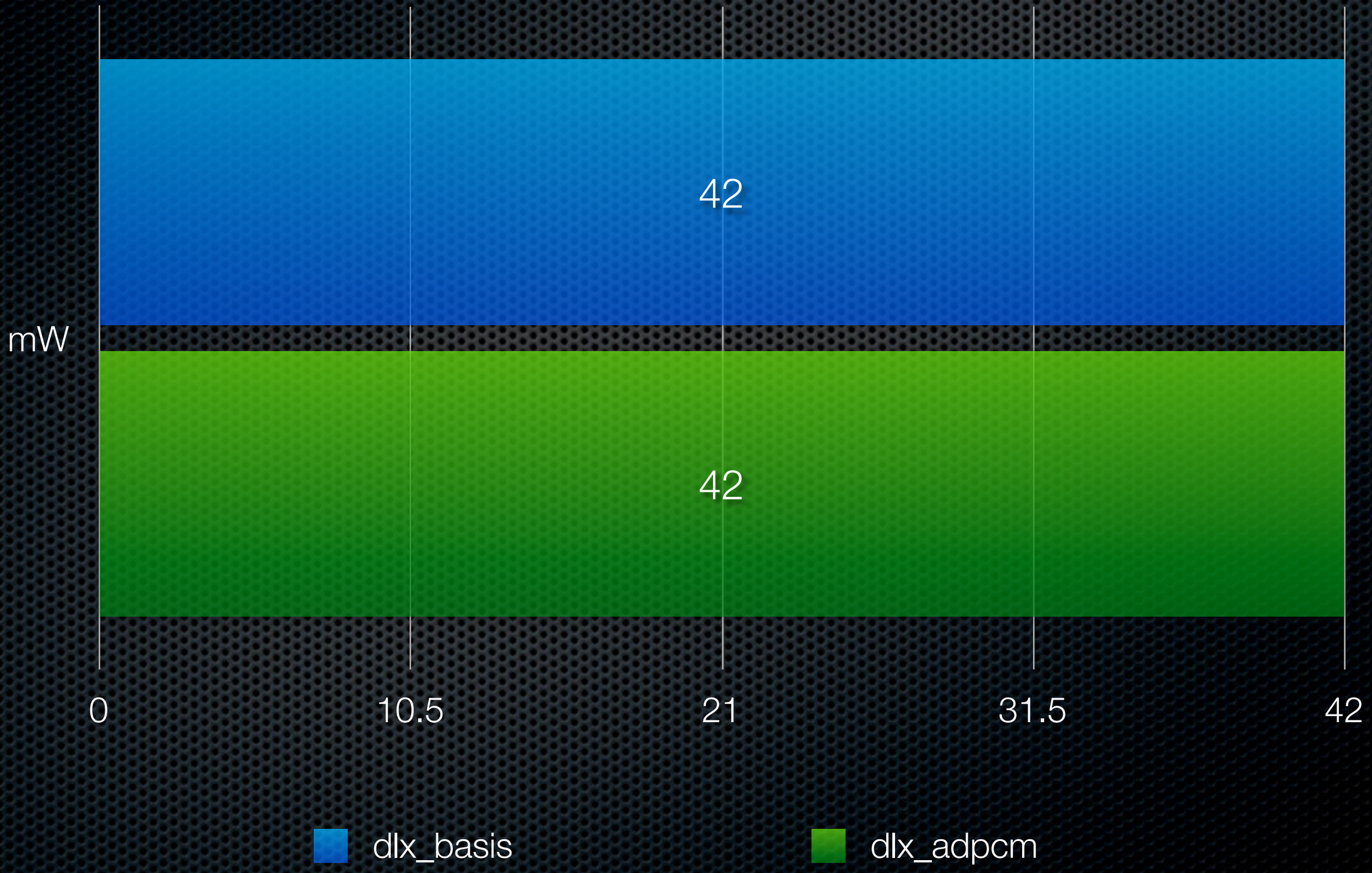
Maximum Frequency



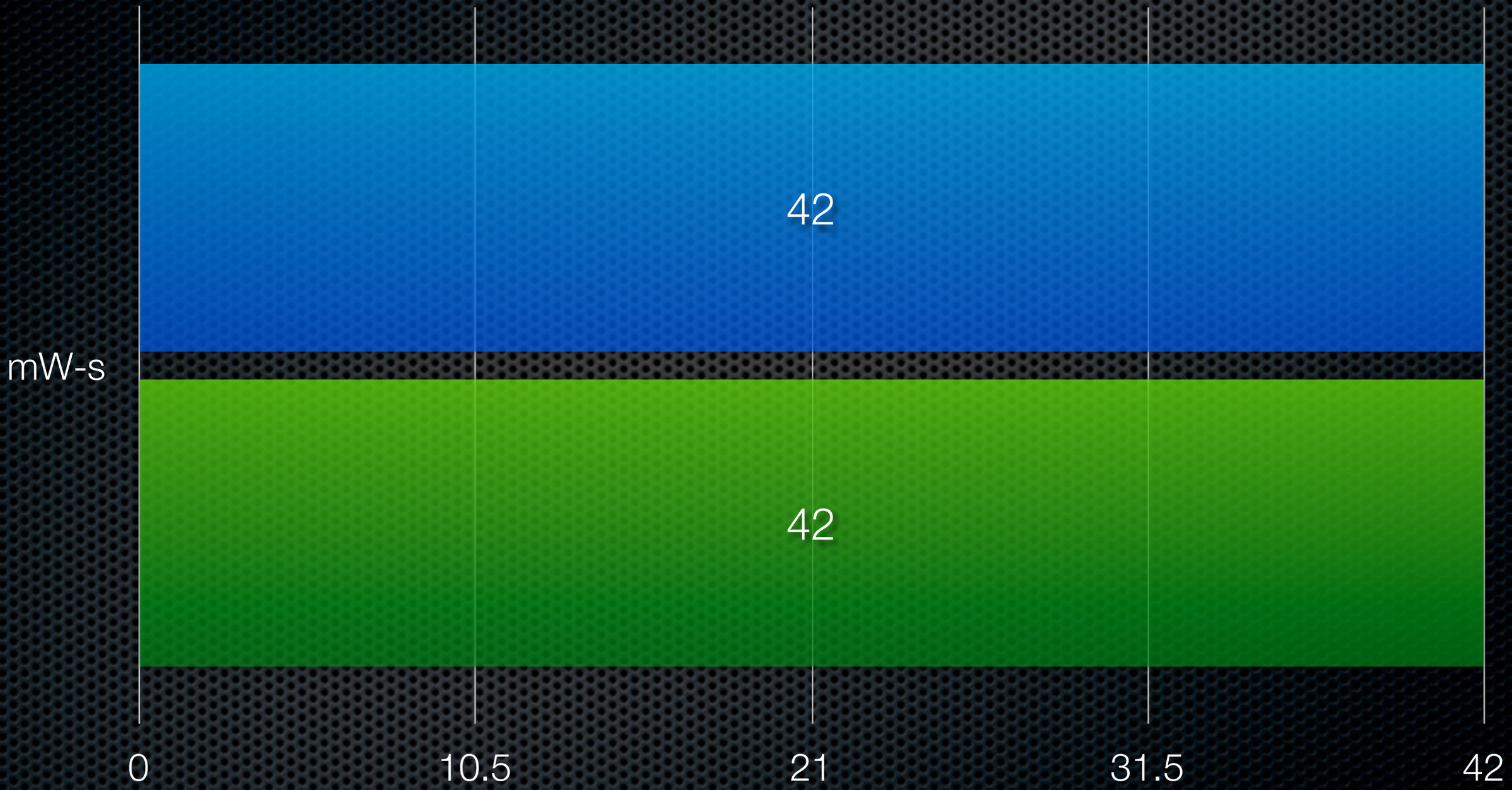
Maximum Frequency Execution Time



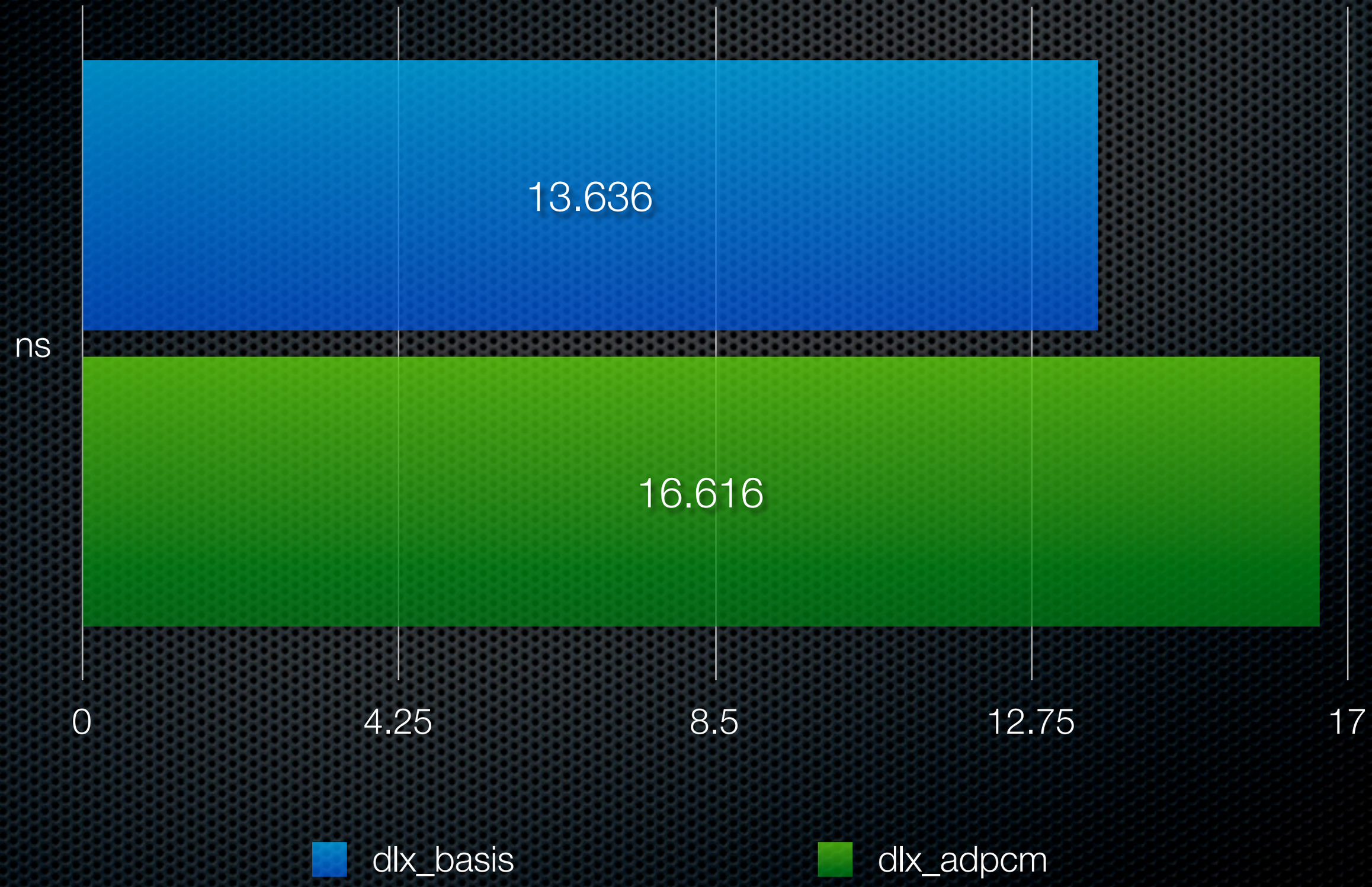
Maximum Frequency Power Consumption



Maximum Frequency Energy Consumption



Critical Path



II Area

Identifying unnecessary Instructions and Resources

- addu, mult, multu, div, divu, xori, srl, srli, slti, sgti,
- slei, sgei, seqi, lh, lhu, sb, sh, beqz, jalr, MOD, modu,
- sltu. sgtu. sleu, sgeu

- MULT0, DIV0

Benchmarks

Area

Slices

2721

2306

0

700

1400

2100

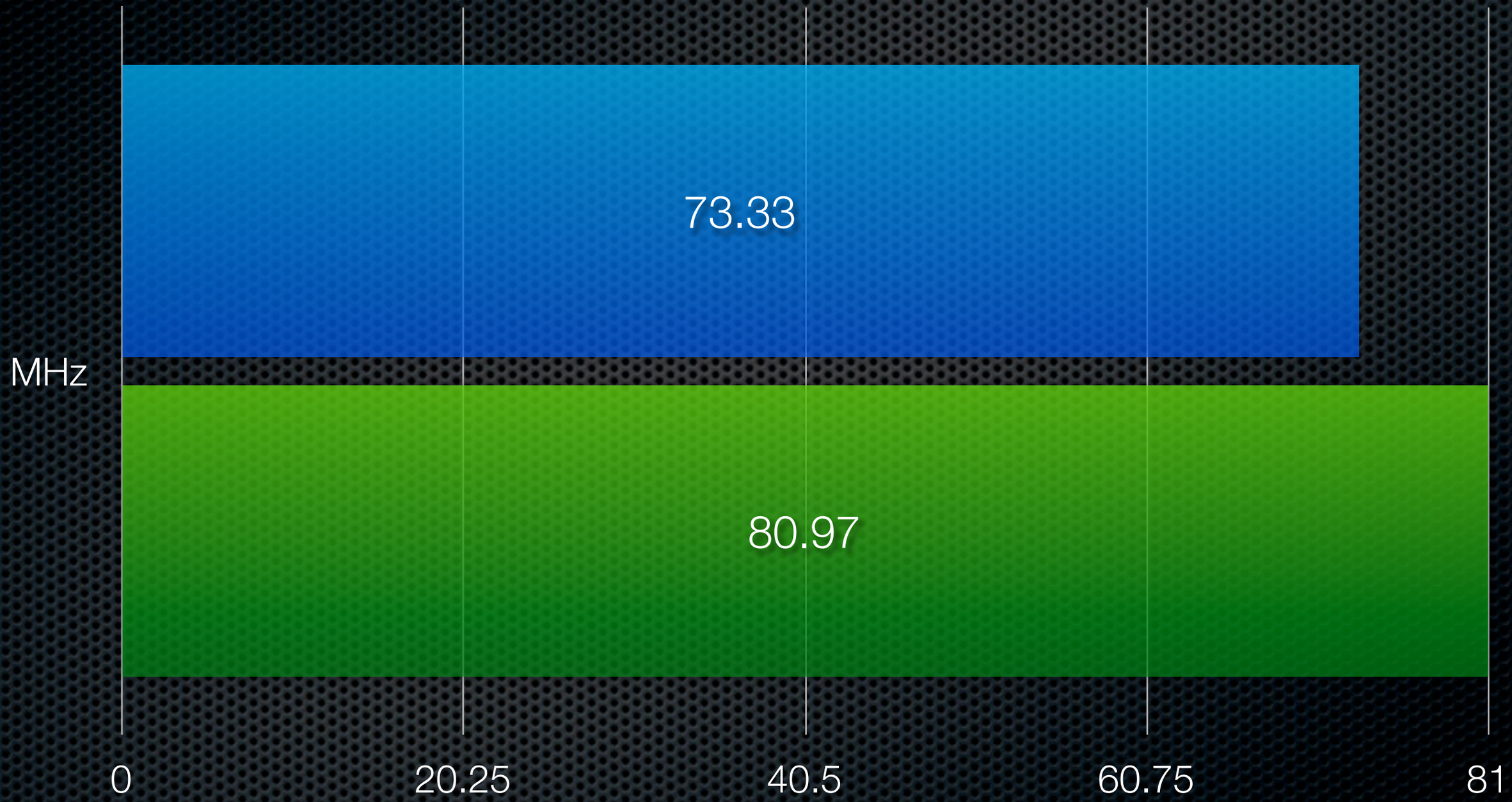
2800

15.25% improvement

dlx_basis

dlx_adpcm_min

Maximum Frequency



Critical Path