Summary

• Using EFLX Embedded FPGA to offload some DSP algorithms takes less clock cycles and consumes less energy than an ARM Cortex M4F for those functions
  ▪ EFLX has higher data throughput than ARM Cortex M4F across all frequencies
    • ARM Cortex M4F requires ~ 17x-31x more clocks than EFLX to perform the same function
  ▪ ARM Cortex M4F consumes ~1.5x – 4.75x energy over EFLX for same computations
  ▪ Actual ARM Cortex M4F power will be higher due to instruction fetch and memory accesses

• Total Energy to perform computation on EFLX is frequency independent
  ▪ Leakage energy contribution at low frequencies can be eliminated by power gating EFLX between computation cycles

• Data throughput of EFLX is fully proportional to EFLX frequency (e.g. no cache misses, etc)

• 5 tap FIR filters and single stage Biquad filters are examples of DSP algorithms that can yield these kind of results

• EFLX numbers based on TSMC 40ULP and ARM Cortex numbers are based on TSMC 40G process (both are comparable in dynamic power)
What is an Embedded FPGA?

Typical FPGA Chip
(Altera Cyclone iV)

- Integrated PCI Express block
- High-speed SERDES up to 3.125 Gbps
- Dedicated multi-purpose PLLs (MPLLs) and PLLs

General Purpose I/O’s

Embedded FPGA

- No High Speed SERDES, PLLs, hard IP bus interfaces, GPIO buffers

Goes on a PCB

Integrated in a chip

flexlogix Technologies, Inc.
Energy Basics

• Battery Life
  - Battery capacity is the maximum amount of energy that can be extracted from the battery under certain specified loads
    - Usually expressed in mA Hours
    - Can also be expressed as energy - Joules (e.g. 1 Joule = 1 Watt Sec)
  - Battery life is the battery capacity divided by the device power
    - Expressed in hours

• Duty cycle
  - Mobile devices run off batteries
  - Mobile devices consume power in short bursts when performing a function
    - Expressed in Joules
  - The shorter the time to perform a function the less energy used
  - The less energy used the less battery drain
  - % Battery Drain is the Total Device Energy used divided by the Battery Energy

- Each shaded box represents the burst energy of a device performing a specific function
- Total device energy is the sum of all the burst energy
  
  **Example:**
  
  Total Device Energy = \((P1 \times (T2-T1)) + (P2 \times (T4-T3))\) = Joules
Power/Energy Details

• **Total Power Formula**
  - $P_{\text{Total}} = P_{\text{Dynamic}} + P_{\text{Leakage}}$
  - Expressed in Watts

• **Dynamic Power**
  - Switching power
  - Toggling signals drawing charge from supply rails
  - Frequency dependent

• **Leakage power**
  - Power supplied and no signals toggling
  - Caused by leakage power
  - Not frequency dependent

• **Energy**
  - Energy = $P_{\text{Dynamic}} \times \text{Time}$
  - Expressed in Joules
  - Also expressed in uW/MHz

\[
P_{\text{avg}} = n \cdot a_{\text{avg}} \cdot f \cdot \frac{1}{2} c_{\text{avg}} V_{dd}^2
\]

- $n$ = number of gates
- $a$ = activity factor (probability of switching on any particular clock period)
# What consumes power on an FPGA chip?

## Power Consumers

<table>
<thead>
<tr>
<th>Function</th>
<th>ASIC</th>
<th>Discrete FPGA</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>Logic</td>
<td>CLBs/LUTs</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Wires</td>
<td>Interconnect fabric</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>I/O</td>
<td>CMOS I/Os</td>
<td>I/O Block + high speed transceivers (e.g. Gb/s)</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>Configuration</td>
<td>None</td>
<td>Configuration SRAM</td>
<td>Static</td>
</tr>
</tbody>
</table>

### Dynamic Power Breakdown in FPGAs

- Interconnect: 10%
- Logic: 60%
- I/O: 38%
- Clocking: 34%

*Does not include High Speed Transceivers*

### Static Power Breakdown in FPGAs

- Interconnect MUXes: 38%
- LUTs: 34%
- Other: 16%
- Configuration SRAM: 12%

*Source: Reconfigurable Logic: Architecture, Tools, and Applications*  
Pierre-Emmanuel Gaillardon
**EFLX embedded FPGA is more power efficient**

- **EFLX is an embedded FPGA**
- **Same process node as rest of SOC/MCU so does not consume more power due to different process**
- **Denser interconnect fabric reduces area and power**
- **No high speed transceivers that consume significant power**
- **Fine grain clock gating at the RBB and Tile levels reduces dynamic power consumption**
  - Unused RBBs are clock gated by compiler
  - Used RBBs in idle can be clock gated by design
- **Power gating at the fabric level**
  - Dynamic
  - No power gating frequency restrictions
  - Reduces leakage power by ~36x when gated off

### Power Gating @ Fabric Level

#### Power Consumers

<table>
<thead>
<tr>
<th>Function</th>
<th>ASIC</th>
<th>Discrete FPGA</th>
<th><strong>EFLX Embedded FPGA</strong></th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>Logic</td>
<td>CLBs/LUTs</td>
<td>RBBS/LUTs</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Wires</td>
<td>Interconnect fabric</td>
<td>Denser Interconnect fabric</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>I/O</td>
<td>CMOS I/Os</td>
<td>I/O Block + high speed transceivers (e.g. Gb/s)</td>
<td>I/O Block, CMOS I/Os, no high speed transceivers</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>Configuration</td>
<td>None</td>
<td>Configuration Cells</td>
<td>Configuration Cells</td>
<td>Static</td>
</tr>
</tbody>
</table>

#### Clock Gating @ RBB & Tile Level

![Clock Gating Diagram]

#### EFLX Fabric

![EFLX Fabric Diagram]
DSP on M4 vs EFLX Power Profile

**DSP Function Executing in M4**
- ARM Cortex M4
- FLASH
- SRAM
- CACHE
- External Memory
- Memory Controller

DSP computation includes memory accesses, instruction fetches and read and writes to SRAM

**DSP Function Executing in EFLX**
- ARM Cortex M4
- CACHE
- FLASH
- SRAM
- AXI Bus
- External Memory
- Memory Controller
- DMA Bus Master
- EFLX DSP

Saving Power By Offloading DSP function with EFLX

DSP on EFLX FPGA can stream data from/to memory directly without going through AXI fabric

Shaded blocks indicate consuming power
Unshaded blocks can be power gated
**EFLX 100 Overview**

<table>
<thead>
<tr>
<th>Name</th>
<th>EFLX-100 FPGA Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>TSMC 40nm ULP CMOS</td>
</tr>
<tr>
<td>Metal Utilization</td>
<td>5 metal layers</td>
</tr>
<tr>
<td>Nominal Supply Voltage (V)</td>
<td>0.9V &amp; 1.1V</td>
</tr>
<tr>
<td>Junction Temperature (°C)</td>
<td>−40 to 125</td>
</tr>
<tr>
<td>Leakage Power (μW)</td>
<td>Deep Sleep Mode 0.5μW with EFLX-100 core and eHVT Bit Cell</td>
</tr>
<tr>
<td></td>
<td>Sleep Mode 1.5μW (at 85°C, 0.9V TT)</td>
</tr>
<tr>
<td>16-bit Counter Frequency (MHz)</td>
<td>110-270 MHz depending on VT/Vdd chosen (TT, 85°C, 0.9 or 1.1V)</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>0.13</td>
</tr>
<tr>
<td>Clock I/O</td>
<td>1 to 8</td>
</tr>
<tr>
<td>Data I/O</td>
<td>152 inputs and 152 outputs</td>
</tr>
<tr>
<td>Total LUTs</td>
<td>Logic Core 120, DSP Core 88</td>
</tr>
<tr>
<td>RBB LUTs</td>
<td>30, 22</td>
</tr>
<tr>
<td>DSP MACs</td>
<td>0, 2</td>
</tr>
<tr>
<td>Array</td>
<td>1×1 to 5×5</td>
</tr>
<tr>
<td>LUT Utilization</td>
<td>&gt;90%</td>
</tr>
</tbody>
</table>

RBB = Reconfigurable Building Blocks
RBB Architecture

**Logic RBB**
- 4 4-input LUTs
- 8 Flip Flops
- Carry chain
- Selectable clocks

**DSP RBB**
- MAC
- 22-bit pre adder
- 22x22 multiplier
- 48-bit post adder (accumulator)
EFLX-100 I/O RBB

Inputs and outputs can be pass-through or flopped
Analysis

• Compare power and energy of running DSP filter algorithms on EFLX FPGA versus a ARM Cortex M4F

• Using TSMC 40ULP eHVT and SVT process nodes for EFLX and TSMC 40G process node for ARM Cortex M4
  ▪ The assumption is that the dynamic power of 40ULP and 40G is similar

• DSP algorithms used were a 5 tap FIR filter and a single stage BIQUAD filter
  ▪ Used 16-bit coefficients and 16-bit/32-bit data for FIR and 16-bit data for Biquad
  ▪ For EFLX, 32-bit data required 2 DSP MACs
    • Each DSP MAC is 22b x 22b + 48b -> 48b
    • 1 clock cycle

• For ARM Cortex M4F, 16-bit and 32-bit data will require same amount of cycles
  ▪ M4F supports 32b x 32b + 64b -> 64b
  ▪ 1 clock cycle
Benchmark Details

- **FIR Filter**
  - Data communications
  - Audio Echo Cancellation
  - Smoothing data

- **Biquad Filter**
  - Audio equalization
  - Motor control

\[ y[n] = \sum_{k=0}^{N-1} h[k] x[n-k] \]

\[ y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + a_1 y[n-1] + a_2 y[n-2] \]

- \( h[4:0], b_{[2:0]} \) & \( a_{[2:1]} \) are coefficients
- \( x[n] \) is data in
- \( y[n] \) is data out

flexlogix Technologies, Inc.
Cortex M4F C-Code for Biquad and FIR Filter

**Single Stage Biquad:**
- 2 memory accesses
- 5 MACs

\[
y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + a_1 y[n-1] + a_2 y[n-2]
\]

\[
xN = *x++;
yN = xN * b0;
yN += xNm1 * b1;
yN += xNm2 * b2;
yN -= yNm1 * a1;
yN -= yNm2 * a2;
\]

Decrement loop counter
Branch

**5 Tap FIR:**
- 13 memory accesses
- 5 MACs

\[
y[n] = \sum_{k=0}^{N-1} h[k] y[n-k]
\]

void fir(q31_t *in, q31_t *out, q31_t *coeffs, int *stateIndexPtr, int filtLen, int blockSize)
{
    int sample;
    int k;
    q31_t sum;
    int stateIndex = *stateIndexPtr;

    for(sample=0; sample < blockSize; sample++)
    {
        state[stateIndex++] = in[sample];
        sum=0;
        for(k=0; k< filtLen; k++)
        {
            sum += coeffs[k] * state[stateIndex];
            stateIndex--;
            if (stateIndex < 0)
            {
                stateIndex = filtLen-1;
            }
        }
        out[sample]=sum;
    }
    *stateIndexPtr = stateIndex;
}

**17.5 Cycles per Data Sample**
**4480 Clock Cycles on Cortex M4F for 256 Samples\(^1\)**

**31.6 Cycles per Data Sample**
**8080 Clock Cycles on Cortex M4F for 256 Samples\(^1\)**

---

**Note 1.** Source: [http://www dspconcepts com - DSPC_Benchmarks.pdf](http://www.dspconcepts.com - DSPC_Benchmarks.pdf)
5 TAP FIR High Level RTL Block for EFLX

- Wishbone Interface Plus FIR Filter
- WB I/F includes the coefficients
16-bit FIR Filter RTL Code

```verilog
module FIR_5TAP (CLK, RESETn, DI, CI, DO);

parameter WL_I = 16 ; // Input Wordlength
parameter WL_O = 16 ; // Output Wordlength
parameter TAP = 5 ; // Number of taps
localparam HTF = TAP ; // Number of "symmetric" taps

input   CLK ;
input   RESETn ;
inout signed [WL_I-1:0] DI ;
inout signed [15:0] CI[0:4];

output reg [WL_O-1:0] DO ;

reg signed [WL_I+16-1:0] Mul_w [0:HTF-1];
reg signed [WL_I+16-1:0] Acc_r [0:TAP-1];

reg w0 [15:0];
reg r0 [15:0];
reg r1 [15:0];
reg r2 [15:0];
reg r3 [15:0];
reg r4 [15:0];

reg x1 [15:0];
reg x2 [15:0];
reg x3 [15:0];
reg x4 [15:0];
reg x5 [15:0];

reg y [15:0];

always @(posedge CLK) Mul_w[i] <= DI * CI[i];
end
endgenerate

always @(posedge CLK) Acc_r[i] <= Mul_w[i];
end
endgenerate

always @(posedge CLK) DO <= Acc_r[TAP-1][WL_I+16-1:WL_I+16-WL_0];
end
endmodule
```


Utilizes 5 EFLX DSP MACs with no additional Logic LUTs required
Optimized 32-Bit FIR Filter RTL Code

Utilizes 10 EFLX DSP MACs with only 8 additional Logic LUTs required

Optimizations of the RTL code was done to better fit into the EFLX DSP MAC structure

Splitting the 32 bit data to upper word and lower word
Shifting the Upper word results and adding to lower word
FIR RTL Schematics

16-bit Data, 16-bit coefficients

32-bit Data, 16-bit coefficients

16x16->32

32-bit output

Fits in 3 EFLX DSP Tiles with no additional Logic Tiles required

2 times 16x16->64

32-bit Data, 16-bit coefficients

Fits in 5 EFLX DSP Tiles with no additional Logic Tiles required

32-bit output

48-bit output
Biquad High Level RTL Block in EFLX

- Wishbone Interface Plus Biquad Filter
- WB I/F includes the coefficients
Biquad Filter RTL Code

```verilog
module Biquad IIR STAP (CLK, RESETn, DI, CI, DO);

3 parameter WL_I = 32 ; // Input Wordlength
4 parameter WL_O = 32; // Output Wordlength
5 parameter TAP = 5 ; // Number of taps
6 localparam MTF = TAP ; // Number of "symmetric" taps

7 input CLK ;
8 input [WL_I-1:0] DI ;
9 output reg [WL_O-1:0] DO ;
10 input signed [WL_I+16:1:0] CI[0:4];
11 reg signed [WL_I+16:1:0] Mul_w [0:HTP-1];
12 reg signed [WL_I+16:1:0] Acc_r [0:TAP-1];
13 // We do a transpose form for the feed-forward FIR,
14 // instead of a direct-mapping form (more resource-efficient).
15 // We also do the retiming on the accumulator.
16 always@ (posedge CLK) Mul_w[0] <= DI * CI[0];
17 always@ (posedge CLK) Mul_w[1] <= DI * CI[1];
18 always@ (posedge CLK) Mul_w[2] <= DI * CI[2];
21 // Accumulation part
22 always@ (posedge CLK) Acc_r[4] <= Mul_w[0];
23 always@ (posedge CLK) Acc_r[1] <= Mul_w[1] + Acc_r[0];
27 always@ (posedge CLK) DO <= Acc_r[4] [WL_I+16:1:WL_I+16-WL_O];
28 endmodule
```

Recursion supported.

16-bit data utilizes 5 EFLX DSP MACs with no additional Logic LUTs required.
16-bit Data, 16-bit coefficients

Fits in 3 EFLX DSP Tiles with no additional Logic Tiles required
## EFLX Utilization

<table>
<thead>
<tr>
<th></th>
<th>16-bit FIR</th>
<th>32-bit FIR</th>
<th>16-bit Biquad</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSPs used</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>LUTs used</td>
<td>54</td>
<td>86</td>
<td>56</td>
</tr>
<tr>
<td>DSP tiles used</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Logic tiles-used</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total tiles used</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Total available LUTs</td>
<td>264</td>
<td>440</td>
<td>264</td>
</tr>
<tr>
<td>LUT Utilization</td>
<td>20.45%</td>
<td>19.55%</td>
<td>21.21%</td>
</tr>
</tbody>
</table>

- EFLX can utilize >90% of LUTs available
- Functions chosen only use small percent of LUT resources available
- Low utilization provides extra resources for additional functionality
Steps used to evaluate Energy

• Extract EFLX dynamic and leakage power for the 3 functions using Power/Rail Analysis tool (Cadence Voltus)

• Total energy per function for EFLX is calculated using 256 cycles for EFLX to run the function

• Compare EFLX total energy required to run each function against ARM Cortex M4 for 256 data samples
Methodology Used for Power Analysis

- FIR/BIQUAD RTL
- Synthesis (Synplify_Pro)
- EDIF
- Compile (EFLX Compiler)
- Bitstream
- Test Bench
- EFLX libraries
- Gate Level Simulation (Questa Sim)
- Gate level synthesis and analysis (Voltus)
- Resource Utilization
  - Max Frequency
- Power Results
  - Dynamic Power
  - Leakage Power
## EFLX-100 Energy per DSP Filter Summary

<table>
<thead>
<tr>
<th>Number of active tiles</th>
<th>16-Bit 5 Tap FIR</th>
<th>32-Bit 5 Tap FIR</th>
<th>16-Bit Biquad</th>
<th>16-Bit 5 Tap FIR</th>
<th>32-Bit 5 Tap FIR</th>
<th>16-Bit Biquad</th>
</tr>
</thead>
<tbody>
<tr>
<td>eHVT/eHVT²</td>
<td>3 DSP</td>
<td>5 DSP</td>
<td>3 DSP</td>
<td>3 DSP</td>
<td>5 DSP</td>
<td>3 DSP</td>
</tr>
<tr>
<td>Total Dynamic Power (P_{Dynamic})¹</td>
<td>42μW/MHz</td>
<td>112 μW/MHz</td>
<td>73μW/MHz</td>
<td>54μW/MHz</td>
<td>144 μW/MHz</td>
<td>94μW/MHz</td>
</tr>
<tr>
<td>Total Energy (E_{Total}) for 256 Cycles</td>
<td>10.75nJ</td>
<td>28.67nJ</td>
<td>18.69nJ</td>
<td>13.82nJ</td>
<td>36.86nJ</td>
<td>24.06nJ</td>
</tr>
<tr>
<td>Deep Sleep Leakage (.5uW/tile)</td>
<td>1.5μW</td>
<td>2.5μW</td>
<td>1.5μW</td>
<td>1.5μW</td>
<td>2.5μW</td>
<td>1.5μW</td>
</tr>
<tr>
<td>Total Leakage Power (P_{Leakage})</td>
<td>10.78μW</td>
<td>17.96μW</td>
<td>10.78μW</td>
<td>68.7μW</td>
<td>114.5μW</td>
<td>68.7μW</td>
</tr>
</tbody>
</table>

Notes:
1. Idle blocks were clock gated. Only DSP clock running
2. Operating condition: tt0p9v85c
   - Typical-Typical process corner
   - Voltage: 0.9 V
   - Temperature: 85 degC

- **Total Energy** is the Energy per cycle x Cycles
- Leakage doesn’t contribute to total energy significantly @ higher frequencies (e.g. >5MHz)

Total Power (P_{Total}) = F*P_{Dynamic} + P_{Leakage}

Total Energy (E_{Total}) = P_{Total} * Time_to_perform_function
= (E_{Cycle} * Cycles) + (P_{Leakage} * [Cycles * (1/F)])
Leakage energy is a large contribution to total energy at lower frequencies if not power gated between cycles.

Total energy to perform computation is frequency independent at higher frequencies.

Leakage energy contribution at low frequencies can be eliminated by power gating EFLX between computation cycles.

Duty Cycle = \( \frac{T_{\text{Computation}}}{T_{\text{Computation}} + T_{\text{Deep Sleep}}} \)
DSPs are main contributor of dynamic power for these specific benchmarks (42.1μW/MHz out of 42.4μW/MHz)
ARM Cortex M4 vs EFLX Data Throughput

- EFLX throughput is higher than M4 across all frequencies
- ARM Frequency can be different than EFLX
- ARM Frequency and EFLX Frequency will scale at the same rate due to both being on the same process technology
## EFLX Dynamic Power & Clock Cycles Compared with ARM Cortex M4

<table>
<thead>
<tr>
<th>Function</th>
<th>Dynamic Power</th>
<th>Clock cycles for 256 data samples</th>
<th>Clock Cycle Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARM Cortex M4(^1)</td>
<td>EFLX-100 eHVT/eHVT</td>
<td>EFLX-100 eHVT/SVT</td>
</tr>
<tr>
<td>16-bit 5 Tap FIR</td>
<td>42(\mu)W/MHz</td>
<td>54(\mu)W/MHz</td>
<td>8080</td>
</tr>
<tr>
<td>32-bit 5 tap FIR</td>
<td>112(\mu)W/MHz</td>
<td>144(\mu)W/MHz</td>
<td>8080</td>
</tr>
<tr>
<td>16-bit Biquad</td>
<td>73(\mu)W/MHz</td>
<td>94(\mu)W/MHz</td>
<td>4480</td>
</tr>
</tbody>
</table>

M4 takes ~ 17x -31x more clock cycles to perform the same function

### Notes:
1. Dynamic power computed based on TSMC 40 nm G process @ .9v, 25C (similar C\(\cdot\)VDD\(^2\) as 40ULP)
   (Source: [http://www-micrel.deis.unibo.it](http://www-micrel.deis.unibo.it) - ARM Cortex-M3/M4 Instruction Set & Architecture)
2. Does not include SRAM, FLASH and memory interface power.
EFLX-100 Total Energy Compared with ARM Cortex M4 For 256 Data Samples

<table>
<thead>
<tr>
<th>Function</th>
<th>ARM Cortex M4 Total Energy&lt;sup&gt;1&lt;/sup&gt;</th>
<th>EFLX Total Energy&lt;sup&gt;3&lt;/sup&gt;</th>
<th>M4 Energy delta over EFLX&lt;sup&gt;4&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit 5 Tap FIR</td>
<td>64.64nJ&lt;sup&gt;2&lt;/sup&gt;</td>
<td>10.75nJ, 13.82nJ</td>
<td>4.76x</td>
</tr>
<tr>
<td>32-bit 5 tap FIR</td>
<td>64.64nJ&lt;sup&gt;2&lt;/sup&gt;</td>
<td>28.67nJ, 36.86nJ</td>
<td>1.75x</td>
</tr>
<tr>
<td>16-bit Biquad</td>
<td>35.84nJ&lt;sup&gt;2&lt;/sup&gt;</td>
<td>18.69nJ, 24.06nJ</td>
<td>1.49x</td>
</tr>
</tbody>
</table>

Notes:
1. Total Energy = M4_joules_per_cycle * M4_clock_cycles
   - M4_joules_per_cycle estimated at 8pJ based on TSMC 40 nm G process @.9v, 25C
2. M4 power will be higher due to cache, instruction fetch power
3. External memory accesses are normalized between M4 and EFLX
4. Using EFLX SVT numbers for comparison

- M4 consumes ~1.5x-4.75x more energy than EFLX for the same function
- Higher energy is caused by M4 taking ~17x-31x more clock cycles to perform the same function
Conclusion

• Using EFLX Embedded FPGA to offload some DSP algorithms takes less clock cycles and consumes less energy than an ARM Cortex M4F for those functions
  ▪ EFLX has higher data throughput than ARM Cortex M4F across all frequencies
    • ARM Cortex M4F requires ~ 17x-31x more clocks than EFLX to perform the same function
  ▪ ARM Cortex M4F consumes ~1.5x – 4.75x energy over EFLX for same computations
  ▪ Actual ARM Cortex M4F power will be higher due to instruction fetch and memory accesses

• Total Energy to perform computation on EFLX is frequency independent
  ▪ Leakage energy contribution at low frequencies can be eliminated by power gating EFLX between computation cycles

• Data throughput of EFLX is fully proportional to EFLX frequency (e.g. no cache misses, etc.)

• 5 tap FIR filters and single stage Biquad filters are examples of DSP algorithms that can yield these kind of results