Computing in Space with OpenSPL

Topic 5: Programming DFEs, basics

Oskar Mencer
Georgi Gaydadjiev

Department of Computing
Imperial College London

http://www.doc.ic.ac.uk/~oskar/
http://www.doc.ic.ac.uk/~georgig/

CO405H course page: http://cc.doc.ic.ac.uk/openspl15/
WebIDE: http://openspl.doc.ic.ac.uk
OpenSPL consortium page: http://www.openspl.org

o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk
Overview

- Simple SLIC interface
- Kernels and Managers
- IDE and getting started
CPU ↔ DFE communication in MaxelerOS

Client Application (C, C++, Fortran, etc) → **SLiC Interface** → CPU x86 → CPU memory → Interconnect (PCI Express etc) → DFE

MaxCompiler (MaxJ) → CoreConfiguration.max “.max-file” → *kernels* → *managers*

Load → **MaxRing** → Dataflow Cores → Multiple kernels → LMEM
About SLiC

- **Simple Live** CPU Interface
- Allows CPU software to use DFEs

- **CPU code must**
  - Include `MaxSLiCInterface.h`
  - Include `MaxFile.max` or `MaxFile.h`
  - Link with `libslic.a` and the compiled `maxfile`
Convolution Kernel

- Simple example computation
  \[ z[i] = a \times (y[i-1]+y[i]+y[i+1]) + x[i] \]
- 2 input streams, 1 input scalar, 1 output stream

```java
public class ConvolveKernel extends Kernel {
    private static final DFEType type = dfeFloat(8, 24);
    public ConvolveKernel(KernelParameters parameters) {
        super(parameters);
        DFEVar x = io.input("x", type);
        DFEVar y = io.input("y", type);
        DFEVar a = io.scalarInput("a", type);

        DFEVar conv = stream.offset(y, -1) + y + stream.offset(y, +1);
        DFEVar z = a*conv + x;

        io.output("z", z, type);
    }
}
```
Simple Manager + CPU code

### CPU code (.c)

```c
#include "Convolve.h"
#include "MaxSLiCInterface.h"

int main(void)
{
    const int size = 384;
    int sizeBytes = size * sizeof(float);
    float *x, *y, *z1, *z2;
    int coeff1 = 3, coeff2 = 5;

    printf("Generating data...
");  // Allocate x, y, z of sizeBytes
    // Initialize x, y data
    printf("Convolving on DFE...
");
    Convolve(size, coeff1, x, y, z1);
    Convolve(size, coeff2, x, z1, z2);

    printf("Done.
");
    return 0;
}
```

### Manager (.maxj)

```java
public class ConvolveManager {

    public static void main(String[] args) {
        // Create kernel and manager
        EngineParameters p = new EngineParameters(args);
        Manager m = new Manager(p);
        Kernel k = new ConvolveKernel(m.makeKernelParameters());

        // Set-up kernel I/O to/from CPU
        m.setKernel(k);
        m.setIO(
            link("x", IODestination.CPU),
            link("y", IODestination.CPU),
            link("z", IODestination.CPU));

        // Auto-generate simple SLiC interface
        m.createSLiCinterface();

        m.build();
    }
}
```

---

**SLiC function generated in MaxFile**

```c
void Convolve(int32_t param_N, double inscalar_ConvolveKernel_a,
              const float* instream_x, const float* instream_y,
              float* outstream_z);
```
SLiC basic static

- Use DFE with a single, simple function call
- Any suitable engine will be selected
- After first use, engine will be held until process terminates
- Multiple MaxFiles can be used by one process – each one will get a dedicated engine
- The `createSLiCinterface()` manager call automatically determines a good set of arguments for the SLiC function
  - We will see how to define more complex interfaces later
SLiC levels

• What if we want more control?
  – Exactly which DFE is used
  – Exactly how long the DFE is reserved for
  – If using multiple MaxFiles, should we use 2 engines or share 1?

• SLiC provides three levels of interaction:
  – Basic Static: single function calls
  – Advanced Static: allows you to run multiple actions on a single engine with a single maxfile, maintaining state on and control of the engine
  – Dynamic: Extension of the advanced static interface using dynamically generated objects to add flexibility at run-time, not limited to static compile-time changes, helps with debugging
Kernels and Managers

Host application

Kernels (instantiate the arithmetic structure)

Manager (arrange the data orchestration)
Simple Application Example

```c
int* x, *y;
for (int i = 0; i < DATA_SIZE; i++)
    y[i] = x[i] * x[i] + 30;
```

\[ y_i = x_i \times x_i + 30 \]
Development Process

Host Code (.c)

```
int *x, *y;
for (int i = 0; i < DATA_SIZE; i++)
  y[i] = x[i] * x[i] + 30;
```

MyManager (.maxj)

```
Manager m = new Manager();
Kernel k =
  new MyKernel();
m.setKernel(k);
m.setIO(
  link("x", CPU),
  link("y", CPU));
m.build();
```

MyKernel (.maxj)

```
DFEVar x = io.input("x", dfeInt(32));
DFEVar result = x * x + 30;
io.output("y", result, dfeInt(32));
```
Manager m = new Manager();
Kernel k =
  new MyKernel();
m.setKernel(k);
m.setIO(
  link("x", CPU),
  link("y", DRAM_LINEAR1D));
m.build();

DFEVar x = io.input("x", dfeInt(32));
DFEVar result = x * x + 30;
io.output("y", result, dfeInt(32));
public class MyKernel extends Kernel {

    public MyKernel (KernelParameters parameters) {
        super(parameters);

        DFEVar x = io.input("x", dfeInt(32));

        DFEVar result = x * x + 30;

        io.output("y", result, dfeInt(32));
    }
}

The Full Kernel
Streaming Data through the Kernel

5 4 3 2 1 0

x

0

x

0

+

30

y

30

30
Streaming Data through the Kernel

5 4 3 2 1 0

x
1
x
1
+
31
30
y
30 31

x
1
x
1
+
31
30
y
30 31

x
1
x
1
+
31
30
y
30 31

x
1
x
1
+
31
30
y
30 31

x
1
x
1
+
31
30
y
30 31
Streaming Data through the Kernel
Streaming Data through the Kernel

5 4 3 2 1 0

x

3

x

9

+

39

30

y

30 31 34 39
Streaming Data through the Kernel
Streaming Data through the Kernel
DFE Programming

- **MaxCompiler** – Java-driven dataflow compiler
- **SLiC Interface** – CPU integration
- **MaxelerOS** – optimized DFE <-> CPU link
- **Seamless simulation environment**
• Java program *generates* a MaxFile *when it runs*

1. Compile the Java into .class files
2. Execute the .class file
   - Builds the dataflow graph in memory
   - Generates the hardware .max file
3. Link the generated .max file with your host program
4. Run the host program
   - Host code automatically configures DFE(s) and interacts with them at run-time
MaxIDE - Design
MaxIDE - Build
MaxCompiler gives detailed latency and area annotation back to the programmer inside MaxIDE.

- Evaluate precise effect of code on latency and chip area.

```
27    17.8:  d.Buy = ask.Price <= lowPrice & order_book.securityId == secId;
28    6.4:  d.Sell = bid.Price >= highPrice & order_book.securityId == secId;
30  
31
32
```

12.8ns + 6.4ns = 19.2ns (total compute latency)
Debugging

- MaxCompiler Simulation
  - Simulate a complete MaxCard/Host system
  - Cycle-accurate
  - Bit-accurate
  - ≈100x faster than circuit simulation

- MaxDebug Hardware Debugging
  - See the status of streams in the DFE
Summary

• SLIC interface allows convenient calls from the host program

• Kernels and Managers help designers focus on the right aspect of the algorithm
  – Kernels for the computing in space
  – Managers for the data orchestration in space

• MaxIDE is a fully integrated Eclipse based environment allowing design, simulation and build of DFE engines