

Handel-C as a Backend Compiler for Synthesizing Reconfigurable Hardware from Functional Specification

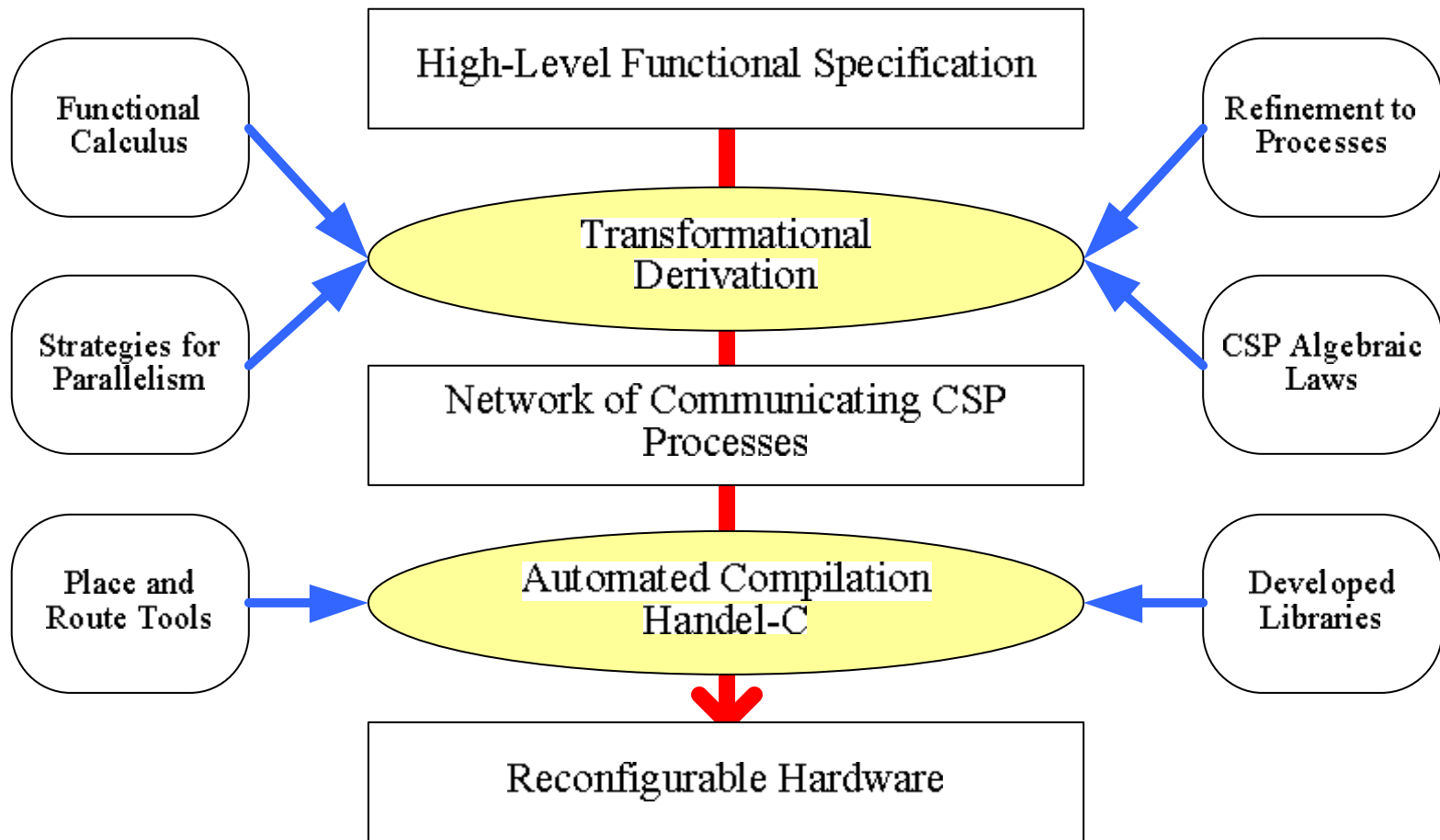
Issam W. Damaj

Ph.D. M.Eng B.Eng MIEEEE MIEE

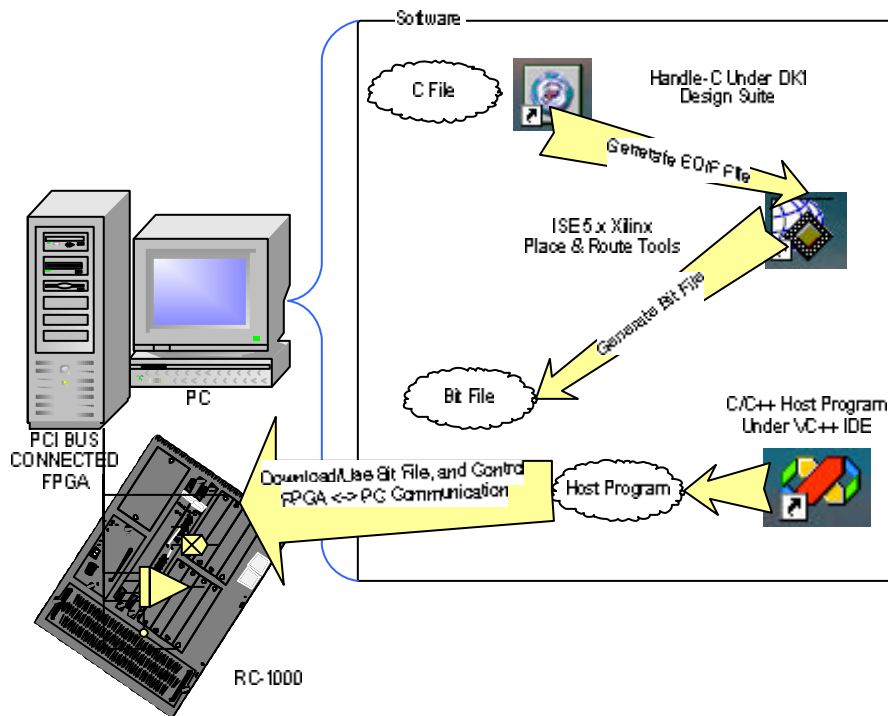
American University of Beirut

Electrical and Computer Engineering Department

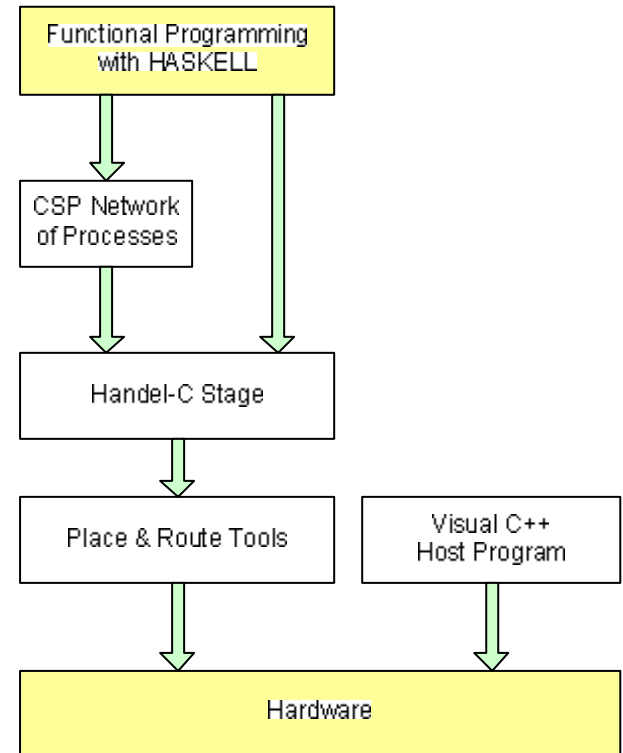
Transformational Derivation



RDM Compilation Steps



Hardware compilation steps.

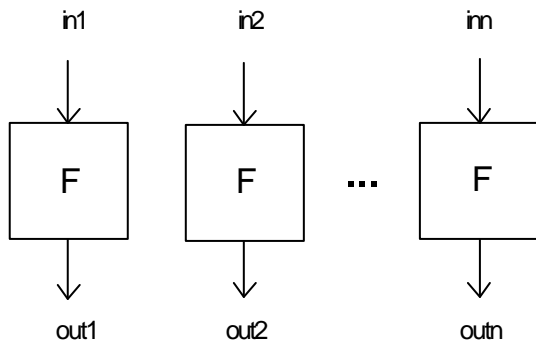


The RDM program development.

High-Order Functions

- Many algorithms can be built from components which are instances of some more general scheme.
- Map applies a function to a list of items. Thus, in the functional setting:
 $map\ f\ [x_1\ x_2\ \dots\ x_n] = [f(x_1)\ f(x_2)\ \dots\ f(x_n)]$
- Refining to CSP:

$$VMAP_n(F) =_{i=1}^{i=n} F[in_i/in, out_i/out,]$$



```
macro proc VMap(in, out, n, F){  
  par (i = 0; i < n ; i++){  
    F(in.elements[i], out.elements[i]);  
  }  
}
```

A data parallel processes visualization of map

$VMAP_n(F)$

High-Order Functions

- The high-order-function *zipWith* is used to zip two lists (taking one element from each list) with a certain operation. The high-order-function *zipWith* is specified as follows:

$$\text{zipWith} :: (A \rightarrow B \rightarrow C) \rightarrow [A] \rightarrow [B] \rightarrow [C]$$

$$\text{zipWith } (\oplus) [x_1, x_2, \dots, x_n] [y_1, y_2, \dots, y_n]$$

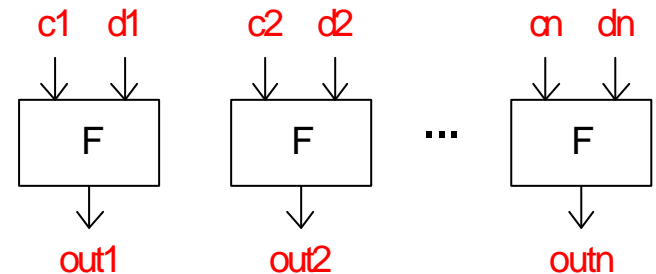
$$= [x_1 \oplus y_1, x_2 \oplus y_2, \dots, x_n \oplus y_n]$$

- To implement the data parallel version of this high-order-function we refine it to a process VZIP:

$$\text{vzip } (\oplus) :: [A]_n \rightarrow [B]_n \rightarrow [C]_n$$

$$\text{VZIP}(\oplus) =_{i=1}^n \parallel F[\text{out}_i / \text{out}, c_i / \text{in}_1, d_i / \text{in}_2]$$

```
macro proc VZipWith( n, in1 , in2, out, F){
    par( i = 0; i < n; Index++){
        F(in1.elements[i], in2.elements[i], out.elements[i]); } }
```



Process VZIPn(F), where F is the refinement of operator \oplus

Advantages of Implementing Cryptographic Algorithms in *FPGAs*

- **Algorithm Agility:** This term refers to the switching of cryptographic algorithms during operation.
 - **Algorithm Upload:** It is perceivable that *FPGAs* could be upgraded with a new encryption algorithm that did not exist (or was not standardized) at design time.
 - **Algorithm Modification:** The modification of a standardized security algorithm is possible, e.g. changing the mode of operation.
 - **Architecture Efficiency:** In certain cases, a hardware architecture can be much more efficient if it is designed with a specific efficient implementation.
- **Throughput:** Although typically slower than an *ASIC* implementations, *FPGA* implementations have the potential of running substantially faster than software implementations.

Simple Example: Rijndael Single Round

```

SingleRound :: State -> State -> State
SingleRound stateIn sKeys =
  AddRoundKey ((mixColumns.shiftRows.subBytes) stateIn) sKeys
  
```

```

SingleRound :: [ [Int8]_4 ]_4 -> [ [Int8]_4 ]_4 -> [ [Int8]_4 ]_4
SingleRound ⊆ SINGLEROUND
  
```

```

SINGLEROUND = (SUBBYTES >> SHIFTRAWS >> MIXCOLUMNS) || ADDROUNDKEY
  
```

```

macro proc SingleRound
  (vovStateIn, vovKStateIn, vovStateout) {
  
```

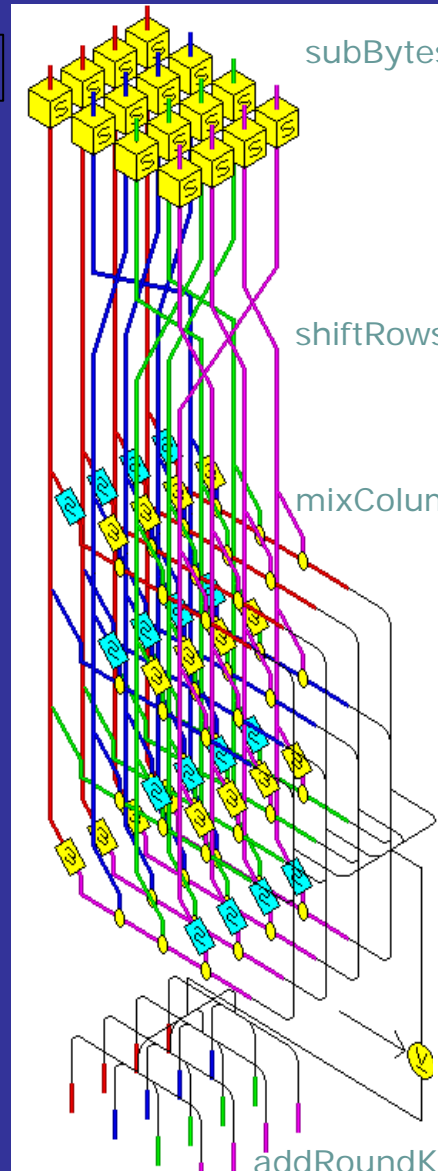
```

  bar{
    SubBytes (vovStateIn, sosout1);
    ShiftRows (sosout1, sovout2);
    MixColumns (sovout2, vovout3);
    AddRoundKey (vovout3, vovKStateIn, vovStateout);}}
  
```

Functional Specification

CSP

Handel-C





Thank You