IRISA/INRIA – Cairn Team

Energy-Efficient Reconfigurable Computing Architectures

University of Rennes 1 – IRISA/INRIA
Campus de Beaulieu – 35042 Rennes
ENSSAT – 6 rue de kerampont – 22300 Lannion
France
http://www.irisa.fr/cairn

ID.Fix Infrastructure for the design of fixed-point systems

Most of digital image and signal processing algorithms are implemented into fixed-point architectures to satisfy the cost and power consumption constraints associated with embedded systems. The fixed-point conversion process is an issue for the reduction of the time-to-market and tools to automate this phase and to explore the design space are required. The ID.Fix tool based on the compiler infrastructure GECOS allows converting a C source code into a fixed-point C code using ac_fixed data types. The data word-lengths are optimized by minimizing the implementation cost under accuracy constraint. To obtain low optimization time, an analytical approach is used to evaluate the fixed-point computation accuracy. This approach is valid for systems made-up of smooth arithmetic operations.

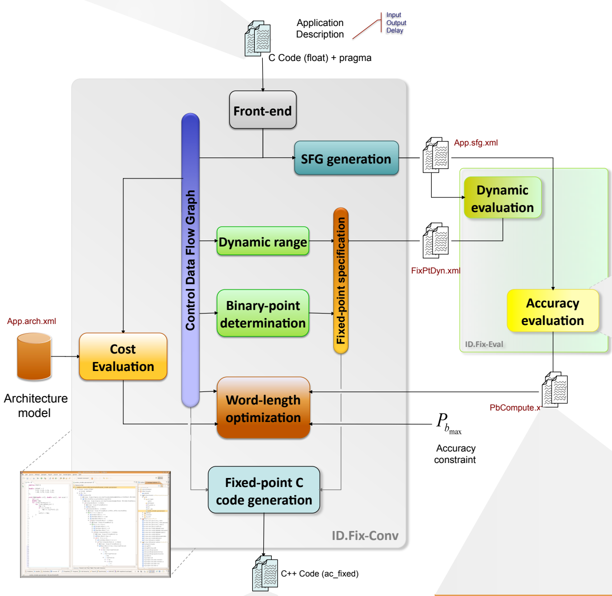
Source code : C code with pragmas

```
float FIR (
#pragma DYNAMIC [-1,1] Wd [16]
float sample
) {

#pragma DELAY
float X[4];

#pragma OUTPUT
float y;
```

Data word-length (bit)
Dynamic range of the input
Output having the accuracy constraint
Delay between tabular elements




```
ac_fixed<16,2,true,AC_TRN,AC_SAT> FIR(
ac_fixed<16,1,true,AC_TRN,AC_SAT> sample)
{
ac_fixed<20,2,true,AC_TRN,AC_SAT> acc;
ac_fixed<16,2,true,AC_TRN,AC_SAT> y;
ac_fixed<16,1,true,AC_TRN,AC_SAT> X[4];
```

Data word-length
Integer part word-length
Saturation mode
Rounding mode
Signe mode

Output code : Fixed-point C code

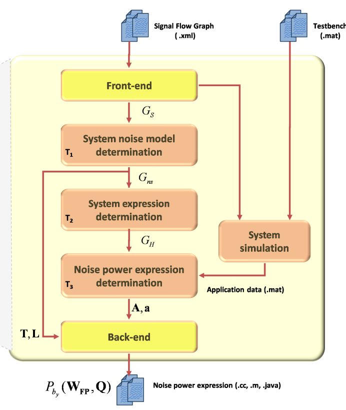
- C++ code with ac data types
  - ✓ Specification of each data with *ac_fixed* data types

Analytical evaluation of the fixed-point accuracy

- **Analytical approach**
  - Significant reduction of the evaluation time compared to fixed-point simulation based approaches
  - Determination of the expression of the global output noise power
    - Technique based on perturbation theory
    - Valid for any system based on smooth operators (+,-,×,÷,…)

$$P_{b_y}(\mathbf{W_{FP}}) = \sum_{i=0}^{N_{ns}-1} K_i.\sigma_i^2(\mathbf{W_{FP}}) + \sum_{i=0}^{N_{ns}-1}\sum_{j=0}^{N_{ns}-1} L_{ij}.\mu_i(\mathbf{W_{FP}}).\mu_j(\mathbf{W_{FP}})$$

Operand Word-length
Gain associated to the variance
Noise source / mean
Noise source / mean
Gain associated to the mean



PbCompute.cc *C code for noise power expression*

```
float CalculRSQB(int** W_FP)
{
...
K[0]=W_FP[107][0] + W_FP[107][1] - W_FP[108][1];
Q[0]=QstepComputation(W_FP[108][1]);

SrcMean[90] = MeanComputation(Q[0],K[0]);
SrcVar[90]  = VarComputation(Q[0],K[0]);
...

...
MeanGain[90]= 0.5348;
VarGain[90] = 0.2345;
...

for(j = 1; j < NB_SRCE_NOISE; j++)
{
   Mean      += SrcMean[i]* MeanGain[j];
   Variance  += SrcVar[i] * VarGain[i];
}

return (10*log10( Variance + Mean * Mean));
}
```

Number of bits eliminated during the cast operation
Computation of the statistical parameters for noise source number 90
Numerical values of the gain (mean and variance) for noise source number 90.
Computation of the global noise mean and variance .

- **T1 : System modeling at noise level**
  - Insertion of noise sources
  - Insertion of data and operator noise model
- **T2 : System expression determination**
  - Cycle dismantling
    - DAG generation
  - Recurrent Equation Determination
    - From DAG to recurrent equation
  - Partial pseudo-transfer Function Determination
    - From recurrent equation to partial transfer function
  - Global Pseudo-transfer Function Determination
    - Computation on Matlab
- **T3 : Noise power expression determination**
  - Computation of the pseudo-impulse response
  - Determination of the gain associated with the mean and the variance
    - Computation on Matlab