A FEC Codec-Processor (ASIP) for Software Defined Radio

SoC Hw / Sw - Co-Design Experience and System Analysis

SDR '04 – Technical Conference
Phoenix, Nov 15th –18th , 2004

Dr. Alfred Blaickner
Carinthia Tech Institute CTI
Email: a.blaickner@cti.ac.at

Overview

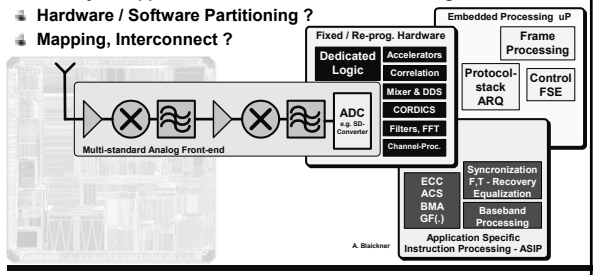
- Motivation
- Introduction
- Error Correction Coding - ECC
  - Principle and algorithms
  - ECC - processor (ASIP)
- System Modeling & Prototyping
  - Model description and bit-true design
  - System modeling - Design flow
  - Implementation platform
- Conclusion

Motivation

- Software Defined Radio – Generic Implementation
  - Multi-standard requirements, Migration: Analog→ Dig.
  - Re-configurable ECC for data transmission
  - Arithmetic cores for dig. Receivers & SDR
  - System modeling and design flow issues
- Real-time Prototyping of Embedded Systems
  - Hardware / Software partitioning and mapping
- R&D and Co-operation
  - EC - SoC- MOBINET (IST-2000-30094), EMS;
  - ESL- Design, HW / SW-Co-design
  - RT- Prototyping – FPGA / DSP
  - Infineon, Synopsys, DTU, … ;

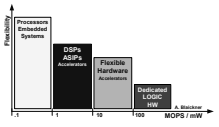
Introduction – System on a Chip (SoC) & Software Radio (SDR)

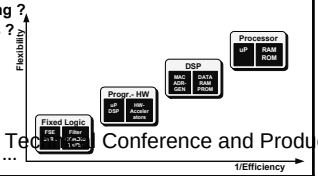
- SoC / SDR – Issues and Constraints
  - May be Applied to Software Defined Radio Design too ...
- Hardware / Software Partitioning ?
- Mapping, Interconnect ?



Introduction – System on a Chip (SoC) & Software Radio (SDR)

- SoC / SDR - Issues and Constraints
- Partitioning – Mapping - Interconnect ?
  - Algorithms/Functions→ HW-Architectures
  - Protocols/Ctrl.- SW→ Functions/Objects
- System on a Chip Design Tradeoffs
  - Specification, Functionality ?
  - Flexibility, HW/SW- partitioning ?
  - Interaction and dependencies ?
  - Interconnection overhead ?
  - Multi-level verification ?
  - Performance / Timing ?
  - Resource estimation ?
  - Power estimation ?
  - Technology, Latency, Cost ? …



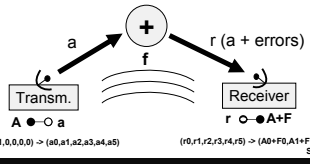


Error Correction Coding - ECC

- Motivation
- Introduction
- Error Correction Coding - ECC
  - Principle and algorithms
  - ECC - processor (ASIP)
- System Modeling & Prototyping
  - Model description and bit-true design
  - System modeling - Design flow
  - Implementation platform
- Conclusion

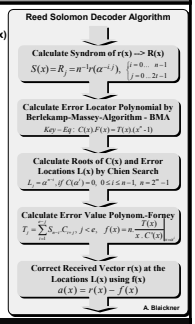
Error Correction Coding - ECC

- Reed-Solomon – Encoding / Decoding
  - Calculated input S → C(x), Generate polynomial for error positions C(x)
  - Polynomial calculations in GF(2^n) supported, common RS(n,k) codes correctable
- Solutions
  - A: RS-decoder (C-Code → SystemC-Compiler)
- Solutions – II
  - B: Processing element / MAC and indexing polynomial calculations
  - Models : MatLab & SystemC, optimize architecture



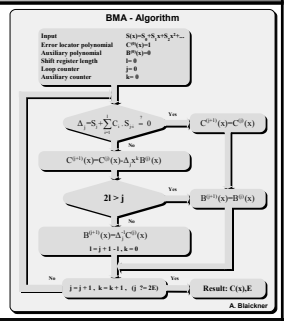
(A0,A1,0,0,0,0) -> (a0,a1,a2,a3,a4,a5)

(r0,r1,r2,r3,r4,r5) -> (A0+F0,A1+F1,F2,F3,F4,F5)
SYNDROME: S



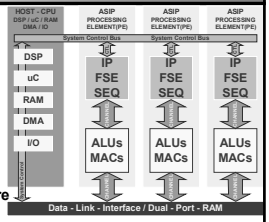
Error Correction Coding - ECC

- Berlekamp-Massey-Algorithm
  - Calculated minimum degree polynomial to generate F(x) out of syndrome S(x)
  - Discrepancy, ?= 0
  - Calculate Cn+1(x) out of Cn(x)
  - 2l>j , l: shift register length
  - Calculate auxiliary polynomial B(x)
  - j>2E , check for max. number of detectable errors
- Most operations are GF-add, GF-multiply, accumulate (convolution) and variable coefficient Indexing



ECC – Processor (ASIP)

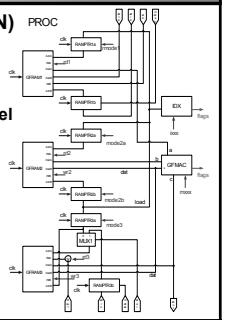
- Dedicated hardware:
  - Full logic implementation → inflexible, *gate graveyard*
  - Less implementation effort and as fast as needed
- More generic solution
  - ASIP, higher implementation effort
  - Flexible - different algorithms
  - Optimizations: op-codes, architecture
- Several solutions available
  - Hardwired – fixed logic vs. general purpose DSP
  - Add HW - accelerators to software based DSP- functions
  - Add flexile data routing and switching



Multiprocessing SoC - Architecture with ASIP accelerators

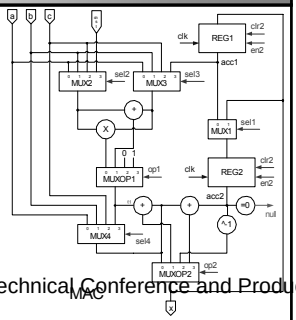
ECC – Processor (ASIP)

- Harvard Type Processing Node (PN)
  - 1 x MAC unit
  - 1 x INDEXER unit
  - 3 x RAM slices
  - Control unit with program ROM, parallel processing
  - Single cycle, two step pipeline (fetch/decode+execute)
- Pipelined
  - RAMs (3x), (inp-inter-out)
  - Pointers (2x)
  - Indexer (1x)
  - MAC-Unit (1x)



ECC – Processor (ASIP)

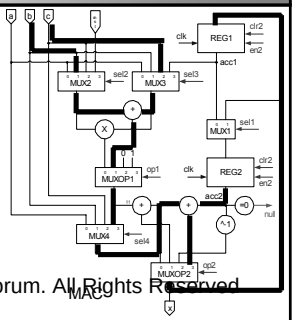
- MAC
- 2 x Accumulators
- 1 x GF-Multiply
- 1 x GF-Invert
- 2 x GF-Add
- 1 x GF-Subtract
- Compare for Zero



MAC

ECC – Processor (ASIP)

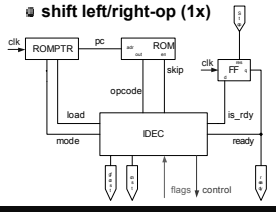
- MAC – Example
- REG1=b+c+REG2
  - sel2=1
  - sel3=2
  - op1=1
  - sel4=3
  - op2=3
  - en2=1
  - Rest: 0 (zero)

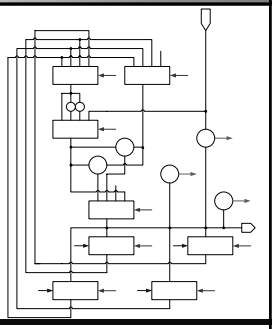


MAC

ECC – Processor (ASIP)

- **Address - Indexer**
  - **index registers (4x)**
  - **Add-op (1x)**
  - **Subtract-op (1x)**
  - **shift left/right-op (1x)**





ECC – Processor (ASIP)

```
+Alternate set:
+
+1               R (ready)
+1                        AAAAAAAA (jmp adr)
+1IIIIIIIIIIII                    CCCCCCCC (cnst)
+1           JJJJ (jump 0..nop/iz/ieq/inz/ineq/in/inn/ing/innq/15..*)
+Regular set:
+                              r r
+                             r m m r
+     i i           mmm m m mm o o m
+ i i s s iiim m mmccs s s so d d o
+ao o e e lllo o eelle e e ed e e d wwrrr
+1p p l l ddddp p nnrrl l l le 2 2 e rrddd
+t2 1 2 1 43212 1 21214 3 2 13 b a 1 32321
:00000000000000011000011000000000000010000  003: ram3(0)=1
:00011001000000000000000000000000001100000  004: ramptr1=m  % calculate discrepancy first
:00011000110000000000000000000000000000000  005: n=1        % n is just loop counter
:00010000000000000000000000001100000000000  006: ramptr3=0
:00000000000000001100000001000000000000001  007: acc1=acc2=ram1(ramptr1) ramptr3++
:10011001100000001000000000001100000000000  008: JMP(n==0) 0x0c % finished with summing up
```

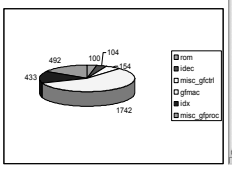

ECC – Processor (ASIP)

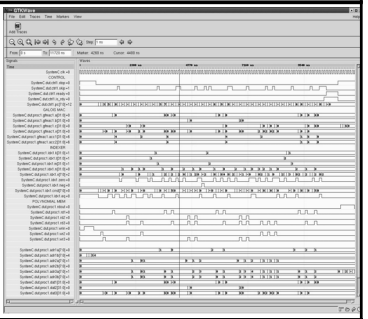
```
INDEXER : SystemC
SC_MODULE(idx)
{
  sc_in_clk clk;
  sc_in<sc_bv<8> > cnst;
  …
  sc_signal<int> k, l, m, n;
  void flags();
  void store();
  SC_CTOR(idx)
  {
      SC_METHOD(flags);
      sensitive << x << cnst;
          …yyz…
      SC_METHOD(store);
      sensitive_pos(clk);
  }
};

#include "idx.h"
  void idx::flags() {
    if (x.read().to_uint()==0) zero.write('1');
    else zero.write('0');
    if (x.read().to_uint()==cnst.read().to_uint()) eq.write('1');
    else eq.write('0');
    ....}
  void idx::store() {
     if (ld1.read().to_uint()==1) k.write(x.read().to_uint());
     ....}
```


ECC – Processor (ASIP)

- **Design Results**
- **Calculates result < 100 cycles**
- **Execution time independent of GF- base**
- **Resources (GE) without RAM**





ECC – Processor (ASIP)

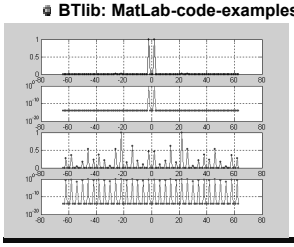
```
function [ResMac,ResMul01]=RSPmac(ClkMac,ClrMac,Clr1,Clr2,Ena1,Ena2,Sel1,Sel2,Sel3,Sel4,Op1,Op2,InA,InB,InC,Const);
global DBGlev;
global N;
persistent ResAcc01; if(isempty(ResAcc01)), ResAcc01=0; end;
....
%:::::::_Init constants ::::::::::::::::::::
const0=0; const1=1;
%=======_Main : GF-MAC
%:::::::_MUX2 ---------------------
Sel=Sel2; InpA=InA; InpB=InB; InpC=InC; InpD=Const;
[ResMux02] = RSPmux41(Sel,InpA,InpB,InpC,InpD);          %MUX2
%:::::::_MUX3 ---------------------
Sel=Sel3; InpA=InA; InpB=InB; InpC=InC; InpD=ResAcc01;
[ResMux03] = RSPmux41(Sel,InpA,InpB,InpC,InpD);          %MUX3
....
%:::::::_idx ---------------
ild1= MaskFcn(in,28,28);
iop1= MaskFcn(in,37,36);
Command=MaskFcn(in,27,24);              %this is the branch handling
switch (Command)
    case {0} %NOP
        mode=bin2dec('01');             %standard case, increment PC
        skip=0;
    case {1} %JMPIZERO
        if (rezc==1)
            skip=1;  %ignore next opcode
            mode=bin2dec('11');         %load PC
        else
            mode=bin2dec('01');         %standard case, increment PC
            skip=0;
    case {2} %JMPIEQ
        if (eq==1)
```

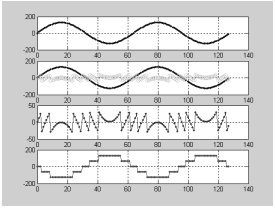

System Modeling & Prototyping

- **Motivation**
- **Introduction**
- **Error Correction Coding - ECC**
  - **Principle and algorithms**
  - **ECC - processor (ASIP)**
- **System Modeling & Prototyping**
  - **Model description and bit-true design**
  - **System modeling - Design flow**
  - **Implementation platform**
- **Conclusion**

System Modeling & Prototyping

- MatLab Bit-true Processing
  - Choice for data path models
  - Accurate arithmetic models
  - BTlib: MatLab-code-examples

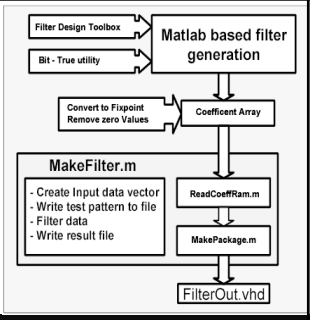




```
BTlib('float2fix',-1:0.5:1,[-1 1],'s',[6 0],'round')
BTlib('add',2.25, 1.0 , 's', [8 1 8 1],'floor')
BTlib('mul',2.25, 1.0, 's', [8 1 8 1], 'floor')
BTlib('shift',2.25, 1, 'l', [8 1 8 1],'u','floor')
BTlib('CheckRange',127.0,[8,2],s)
```

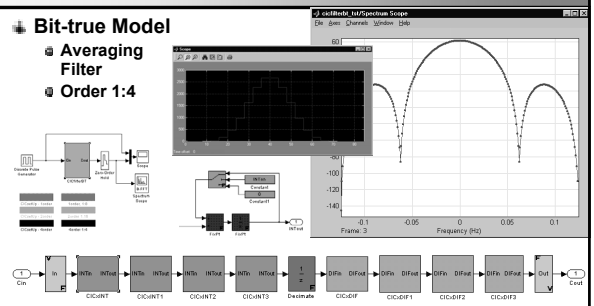

System Modeling & Prototyping

- Design and Pre- Verification in MatLab
- Bit-true / Cycle-accurate Modeling
- Template based VHDL Code-Generation
- Synthesis to the FPGA – Platform & Post-Simulation
- Real-time Test and Verification on the Rapid - Prototyping System – 'PASS'



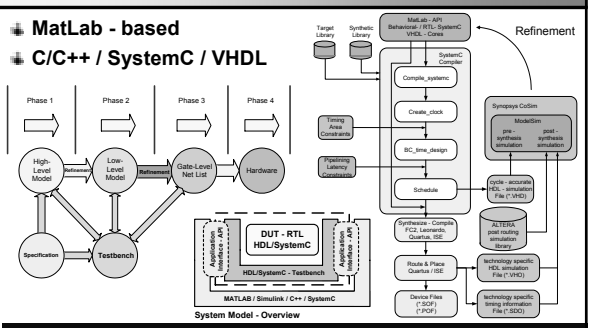
System Modeling & Prototyping

- Bit-true Model
  - Averaging Filter
  - Order 1:4



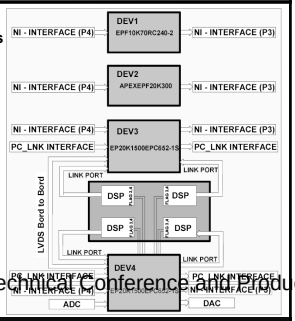
System Modeling & Prototyping

- MatLab - based
- C/C++ / SystemC / VHDL



Implementation Platform FPGA/DSP - based Prototyping

- DSPs used for Test- Vector Generation and Data Post-Analysis
- FPGA – DSP Connection via 40 Mbyte/s Link - Interface
- Visualization of RT-Results in Matlab / C/C++ / SysC



Conclusion

- Introduction – SoC & SDR
- Error Correction Coding - ECC
  - Principle and algorithms
  - ECC - processor (ASIP)
- System Modeling and Prototyping
- Thank you for Attention !