AMI Analysis Using a Proxy Class

DesignCon IBIS Summit
Santa Clara, CA
February 3rd, 2017

Wei-hsing Huang, SPISim
Wei-hsing.Huang@spisim.com
Agenda:

● Background
● Motivation
● Approach: A proxy class
● Analysis/Experiments:
  ○ Stress and consistency tests
  ○ Co-optimization within internal process
  ○ Co-optimization/Simulation with external process
● Summary
● Q & A
Background: Roles in an AMI flow

● AMI simulation flow: [1]
  ○ Simulation platform and AMI models
    ■ Communicate via AMI API defined in IBIS spec.
  ○ Exchange settings via “Reserved parameters”

● Back-channel co-optimization flow: [2][3][4]
  ○ Simulation platform, Protocols, and AMI models
  ○ Models exchange settings via “BCI*” parameters
  ○ Different types of models
    ■ Legacy, hardware protocol backed or others
Motivation: Bridge the gaps

● A “mediator” in the flow
  ○ Between simulator and models, or between different models
  ○ Mainly for development purpose (testing and experiments)

● For model & optimization development:
  ○ Don’t want to wait until BIRD 147.4/Spec. finalized
  ○ My favorite simulator may not support new spec. or get updated
  ○ Want to use existing simulator and models, now
  ○ Need to process data outside simulator and model
  ○ Need to reuse simulator’s post-process functions
Approach: A Proxy Class

- A proxy class: [5]
  - Implements AMI API
  - Called/Loaded by simulator
  - Calls/loads actual models
  - Is a “Man in the middle” [6]
    - Can intercept, modify data
    - Can perform customized flow

- A proxy class is an AMI-compatible .dll(s)/.so(s) which loads actual model’s AMI .dll(s)/.so(s) and does things...
Proxy class code snippet:

```c
#include <windows.h>

typedef long (*amiInit)(double *htInput, long rowSize, long numAggr,
            double sampInt, double bitTime, char *inpPam,
            char **outParm, void **modLPtr, char **message);

typedef long (*amiGetWav)(double *wavData, long wavSize, double *clkTime,
            char **outParm, void *modLPtr);

typedef long (*amiFin)(void *modLPtr);

// Proxy's Init function: called by the simulator and delegate to real model *
IBIS_AMI_API long AMI_Init(double *htInput, long rowSize, long numAggr,
            double sampInt, double bitTime, char *inpPam,
            char **outParm, void **modLMem, char **message) {
    HWODULE dllLibs = LoadLibrary("C:/Temp/SPISimAMI.Tx.dll");
    amiInit ptrInit = (amiInit) GetProcAddress(dllLibs, "AMI_Init");
    long initStatus = (ptrInit)(htInput[0], rowSize, numAggr, sampInt, bitTime,
            inpPam, outParm, modLMem, message);
    FreeLibrary(dllLibs);
    return initStatus;
}
```
Analysis 1: Consistency and stress tests

- Waveform results of first and last loop should be the same
- Monitor resource, memory usage should stay roughly the same
- Important as AMI_GetWave may be called many times for lengthy bits

```c
/* The following is proxy class's getWave function, called by the simulator */
IBIS_AMI_API long AMI_GetWave(double *wavData, long wavSize, double *clkTime,
    char **outParm, void *modPtr) {

    HMODULE dllLibs = LoadLibrary("C:\Temp\SPISimAMI_Tx.dll");
    amiGwAv ptrGwAv = (amiGwAv) GetProcAddress(dllLibs, "AMI_GetWave");
    double *tstData = (double*)calloc(wavSize, sizeof(double));

    // loop for stress and consistency tests
    for (int i = 0; i < 10000; i++) {
        // duplicate test data from original
        memcpy(tstData, wavData, sizeof(double) * wavSize);
        // call model's getWave function
        assert((ptrGwAv)(tstData, wavSize, clkTime, outParm, modPtr) == 1);
    }

    FreeLibrary(dllLibs);
    free(tstData);
    tstData = 0;
    return 1;
}
```

Call `getWave` many times...
Analysis 2: Co-optimization (internal)

- Convolution/FFT is “commutative”
- Use “delta response” in Tx
  - Basically returns same wave_data
- Combine Tx & Rx in Rx’s proxy class
  - Optimization separately or together
  - Combine “Model_Specific” parameters
Analysis 2: code snippet

```c
// Proxy's model has multiple stages
typedef struct ProxyMdl {
    void *txMod1;
    void *txMod2;
};
// My own optimization routine
bool optimize(double *wavData, long wavSize, char **outParm, void *mod1Ptr);

/* The following is proxy class's getWave function, called by the simulator */
IBIS_AMI_API long AMI_GetWave(double *wavData, long wavSize, double *clkTime, char **outParm, void *mod1Ptr) {
    MODULE txDll = loadLibrary("C:/Temp/SPISimAMI_Tx.dll");
    MODULE rxDll = loadLibrary("C:/Temp/SPISimAMI_Rx.dll");
    amiOnAv txOnAv = (amiOnAv) GetProcAddress(txDll, "AMI_GetWave");
    amiOnAv rxOnAv = (amiOnAv) GetProcAddress(rxDll, "AMI_GetWave");
    if (txOnAv && rxOnAv) {
        // Proxy contains multiple stages
        void *txMod1 = (ProxyMdl*)mod1Ptr->txMod1;
        void *txMod2 = (ProxyMdl*)mod1Ptr->txMod2;
        // Optimize TX & RX until converged
        long txStat, rxStat;
        bool converged = false;
        while (!converged) {
            // call getWave of various stages
            txStat = (txOnAv)(wavData, wavSize, clkTime, outParm, txMod1);
            rxStat = (rxOnAv)(wavData, wavSize, clkTime, outParm, rxMod1);
            // call my own optimization routine, update param and check converged
            converged = optimize(wavData, wavSize, outParm, mod1Ptr);
            assert((txStat - 1) && (rxStat - 1));
        }
    } freeLibrary(txDll);
    freeLibrary(rxDll);
    return 1;
}
```
Analysis 3: Co-optimization (external)

- Use simulator’s post-processing to get performance metrics
  - Simulator loads a proxy class
  - This proxy class serves as a “client” \[8\]  
  - Query server and feeds data back to simulator
  - Simulator post-processes data and writes results to disk

- A standalone process (e.g. AMI test driver) loaded a proxy class \[9\]
  - This proxy class serves as a “server” (codes similar to Analysis 2)
  - Use socket, IPC or file to communicate with client
  - Persistent across multiple simulator invocations
  - Client response changes get updated continuously
    - Based on parameters passed by client (via socket), or
    - Based on post-processed results from simulator
Analysis 3: flow diagram

Stand-alone process
(Persistent, as a Server)

Simulator
(Invoked many times, as a Client)

AMI Proxy
Tx getWave
Rx getWave
...

Socket/IPC Communications
Exchange param./wave_data

AMI Proxy
No real models are loaded...

Post-processed results (EH, EW, BER etc)

Update param. Based on simulator’s post-processing results.
Analysis 3: client code snippet

```c
/* The following is proxy class's getWave function, called by the simulator */

IBIS_AMI_API long AMI_GetWave(double *wavData, long wavSize, double *clkTime,
   char **outParm, void *modPtr) {
   // variables declarations, clean-up and error checks omitted below...

   // convert data to buffer to be sent to server
   byte *sendbuf = convDataToBuffer(wavData, wavSize, outParm);

   // Resolve the server address and port
   iResult = getaddinfo(argv[], DEFAULT_PORT, &hints, &result);

   // Attempt to connect to an address until one succeeds
   for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {
      // Create a SOCKET for connecting to server
      ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai protocol);
      // Connect to server.
      iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
      if (iResult == SOCKET_ERROR) {
         freeaddrinfo(result);
         continue;
      }
      break;
   }
   freeaddrinfo(result);

   // Send an initial buffer
   iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);

   // shutdown the connection since no more data will be sent
   iResult = shutdown(ConnectSocket, SD_SEND);

   // Receive data
   byte *recvbuf = recv(ConnectSocket, recvbuflen, 0);

   // cleanup
   closesocket(ConnectSocket);

   // convert receiving buffer data back to waveform for calling simulator
   convBufferToData(recvBuf, &wavData, &wavSize, &outParm);
   return 1;
}
```
Summary:

- A proxy class is useful for AMI development:
  - An AMI .dll(s)/.so(s) calling model’s AMI .dll(s)/.so(s)
    - Can capture, post-processing and redirect waveform data
    - Can intercept calls and perform customized flow
    - Modularized to be independent with simulator and models
    - Can be applied to today’s simulators and models

- Some useful testing/experiments:
  - Consistency and stress tests of AMI models
  - (Back-channel) co-optimization
    - Using internal or external process
  - Can integrate into either simulator or models easily
References:

1. IBIS Spec. V6.1
2. Co-Optimization of SerDes Channels using AMI Modeling, June, 2014, IBIS Summit at San Francisco
3. The Backchannel Crossroads, June, 2014, IBIS Summit at San Francisco
4. BIRD 147.4 Back-channel Support draft 5
5. Proxy design pattern
6. Man in the middle
7. Simulating High-Speed Serial Channels with IBIS-AMI Models (KeySight)
8. Winsock client and server code sample
9. IBIS-AMI test drivers (e.g. from SPISim, SISoft and Cadence or HSpice’s AMICheck)
EDA Expertise in Signal, Power Integrity & Simulation

SPISim is an InSync member.