High-Level Architecture Exploration for MPEG4 Encoder with Custom Parameters

Marius Bonaciu, Aimen Bouchhima, Wassim Youssef,
Xi Chen, Wander Cesario(*), Ahmed Jerraya

TIMA Laboratory, SLS Group, Grenoble, FRANCE

MND, Paris, FRANCE (*)

Outline

- Introduction to High-Level Architecture Exploration
- Flexible Architecture Model for Architecture Exploration
- Flexible Algorithm and Architecture Representation
- High-Level Architecture Exploration Flow for MPEG4 Encoder
- Conclusions

MPEG4 Encoder Applications

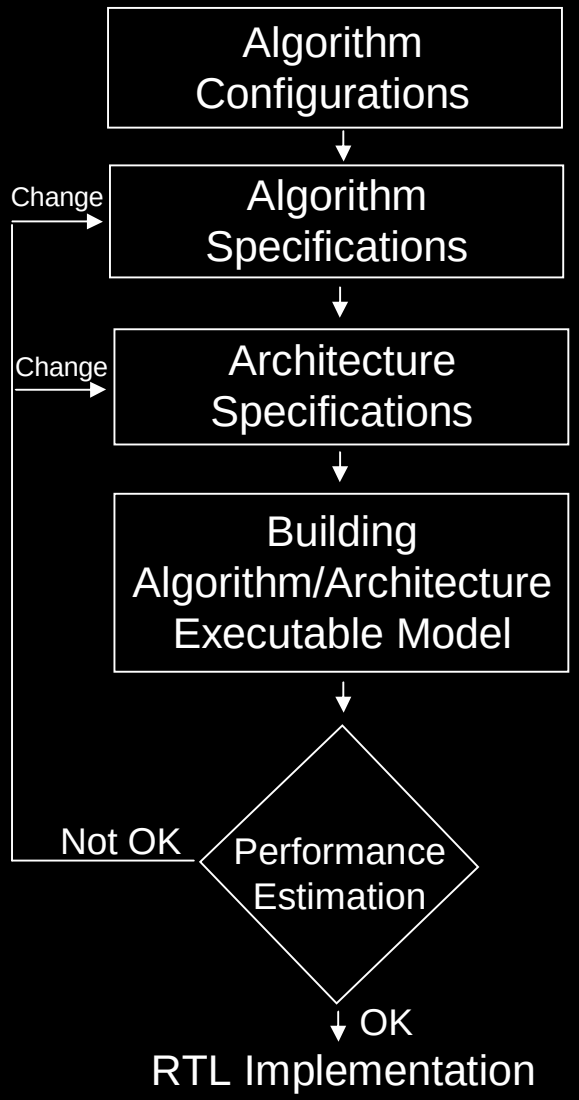
- Many application domains → different configurations, architectures, constraints

Solution Space for MPEG4 Architecture/Algorithm Exploration

- MPEG4 Algorithm Parameters
  - Video resolution
  - Frame_rate
  - Bitrate
  - MotionEstimation precision
  - Motion Search Area
  - Progressive/Interlaced
  - Scene Change Detection
  - Quantization range/type
  - …
- Architecture Parameters
  - Number of CPUs
  - Type of CPUs
  - HW-SW partitioning
  - Communication topology
  - Arbitration type
  - Message size
  - Data width
  - Data transfer latency
  - …

- Finding the optimal solution requires to explore between very large number of solutions
- Implementing the RTL architecture using wrong configurations might “kill” the project

Classical Exploration Flow

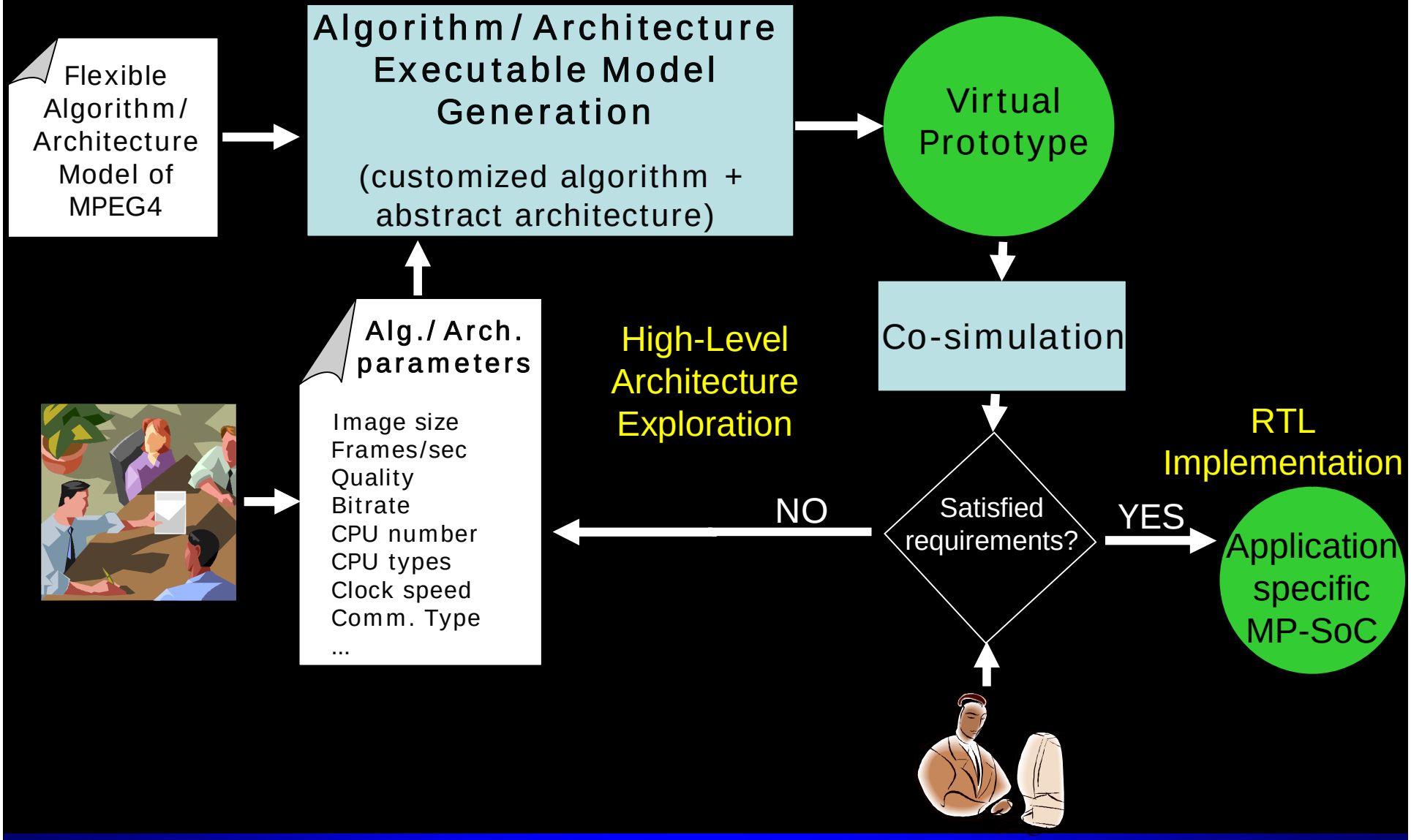


- **Algorithm/Architecture Specifications**
  - What can be changed:
    - parallelism, pipelining
    - mapping
    - data organization
    - communication
- **Building the Algorithm/Arch. Executable Model**
  - Designing manually → fastidious, long time, errors
  - Need re-designing a new Model for every Algorithm/Architecture configurations change
  - Simulation speed depends on the abstraction level
- **Performance estimation precision**
  - Depends on the abstraction level
  - Is a key issue, no matter the abstraction level

Motivation and Objective

- Need to explore large solution space
  - Automatic generation of Executable Algorithm/Architecture Model
    - A unique Flexible Algorithm/Architecture Model of MPEG4 Encoder used to obtain different Algorithm/Architecture Executable Models
      → automatically customize the algorithm & build the abstract architecture
- Need of fast simulation
  - Architecture Exploration at High-Level
    - By ignoring many low-level architecture details, simulation becomes fast
- Need of precise simulation results
  - High precision for the High-Level Architecture Exploration
    - Precise estimations of the computation times
    - Precise estimations of the communication times

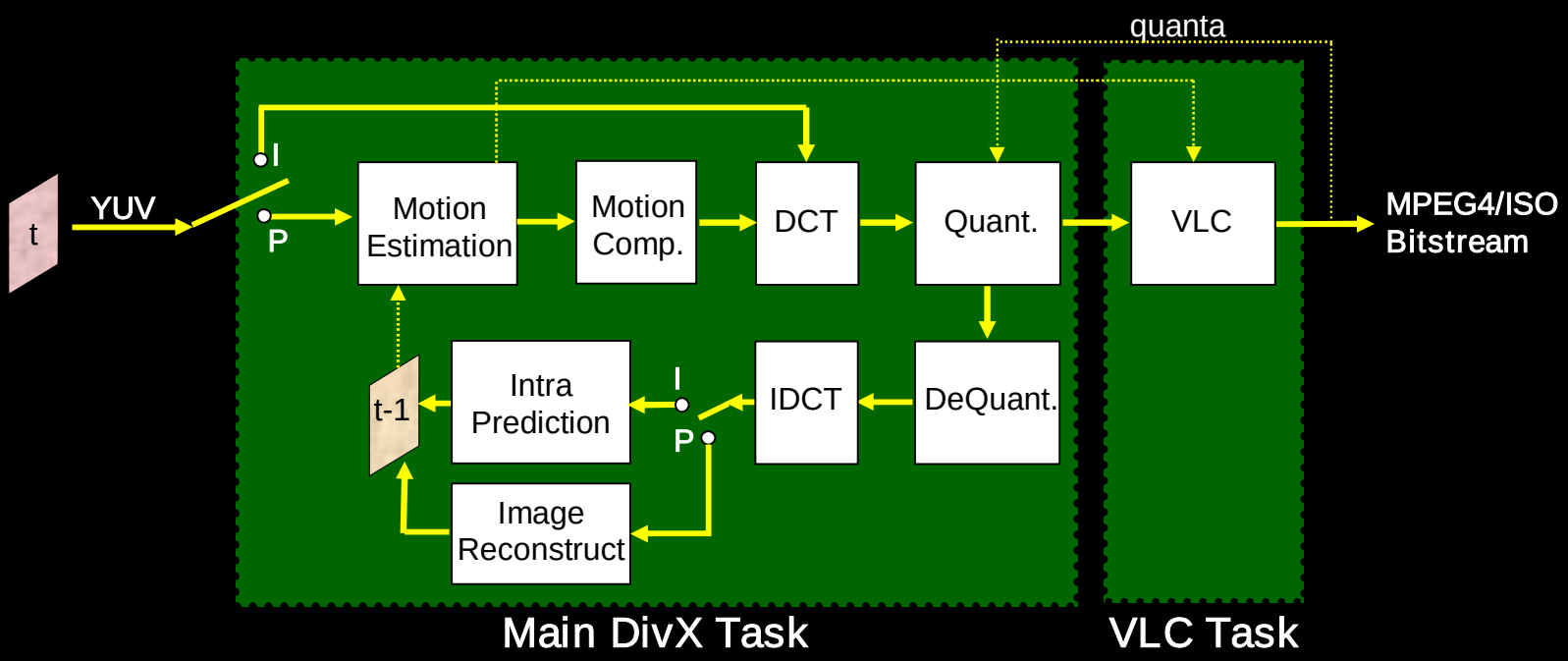
Contribution for High-Level Architecture Exploration



Outline

- Introduction to High-Level Architecture Exploration
- Flexible Architecture Model for Architecture Exploration
- Flexible Algorithm and Architecture Representation
- High-Level Architecture Exploration Flow for MPEG4 Encoder
- Conclusions

MPEG4 Video Encoder Algorithm

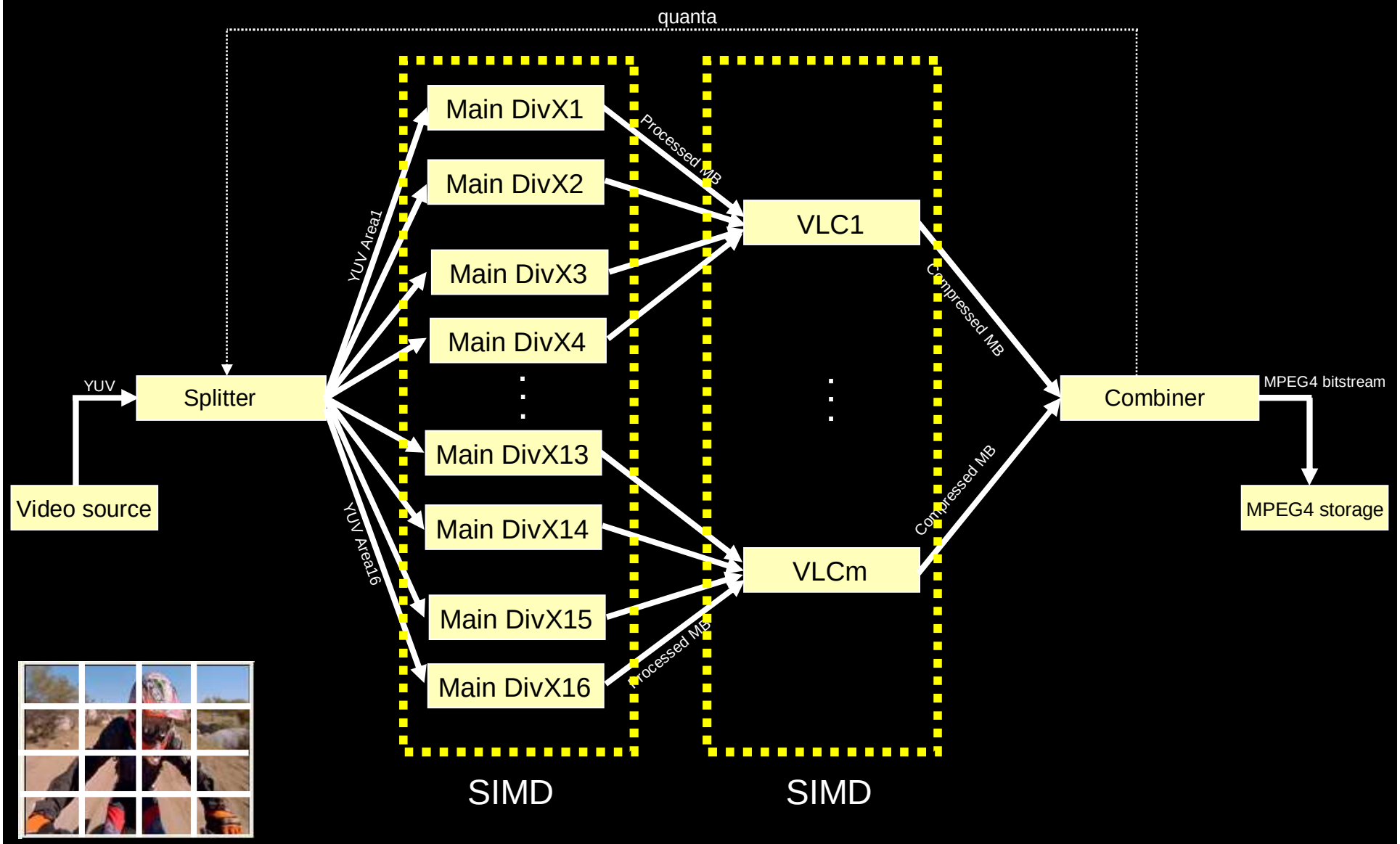


- Compress only the spatio-temporal differences between consecutive frames
- DivX = a popular algorithm implementation of the MPEG4 video compression technology (ISO/IEC 14496-2)

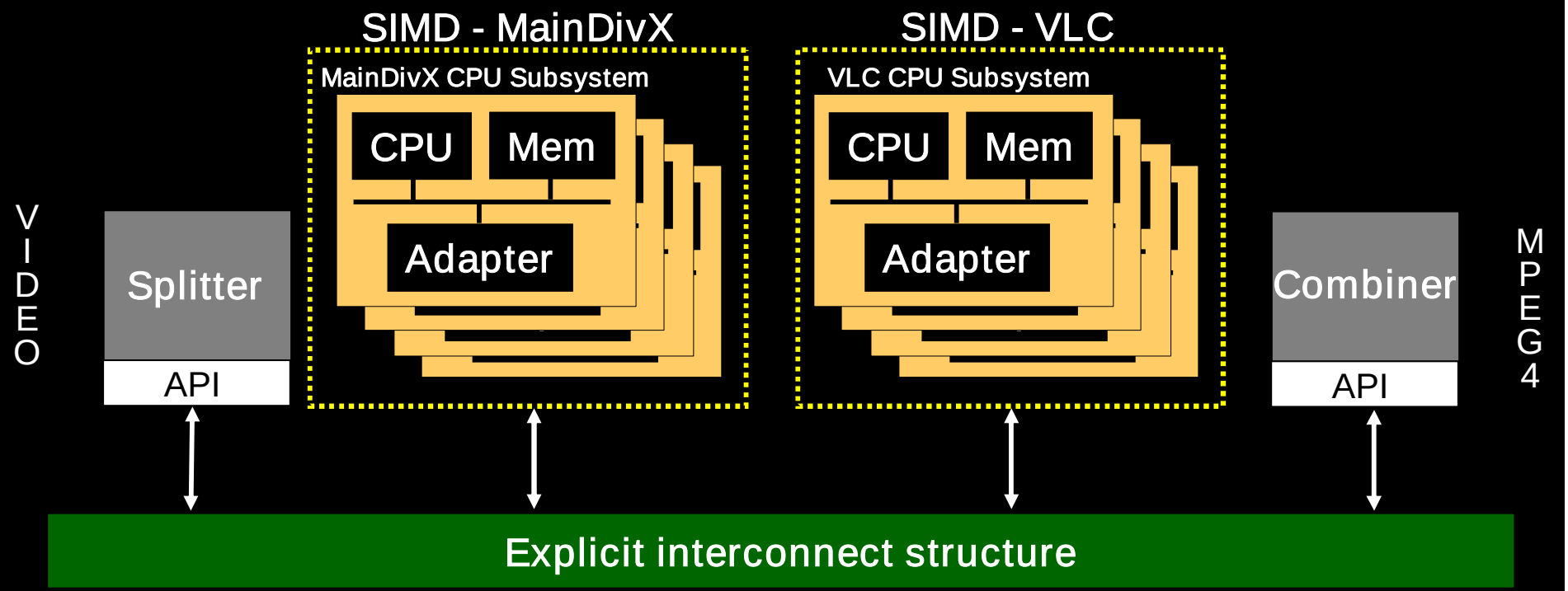
Algorithm Exploration

- **Initial MPEG4 Encoder algorithm**
  - sequential algorithm
  - requires a large amount of computation
- **Initial Exploration Parameters**
  - Video resolution, Frame rate, Bitrate, MotionEstimation precision, Motion Search Area, Progressive/Interlaced, Scene Change Detection, Quantization range, Quantization type
  - Insufficient for multi-processor implementation
- **Need of adding parameters for multi-processor implementation**
  - Insert parallelism and pipeline support
  - Modifying parallelism/pipeline level shouldn't change the code, only the behavior
- **Flexible MPEG4 Encoder algorithm**
  - Supports all the Initial Exploration Parameters, plus Parameters for the level of Parallelism/Pipeline

Flexible Parallel/Pipeline DivX Data Flow



Targeted Abstract Architecture with 2 SIMD

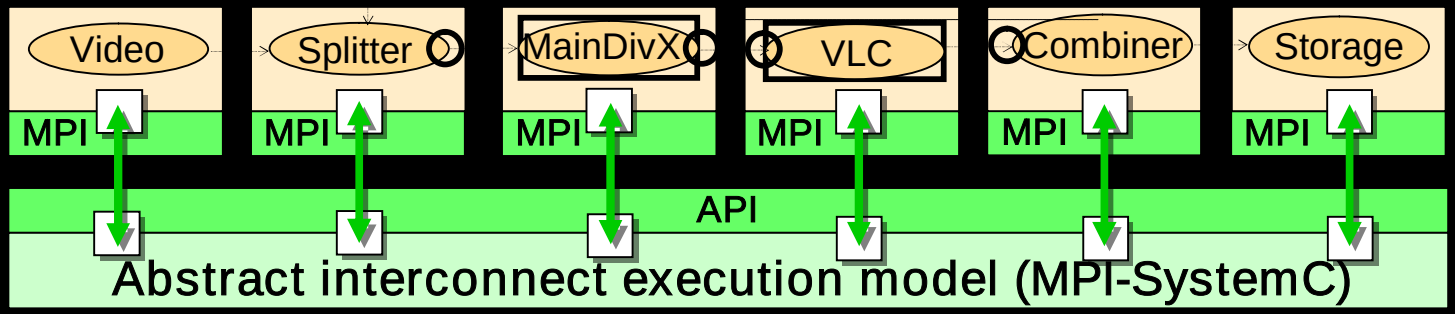


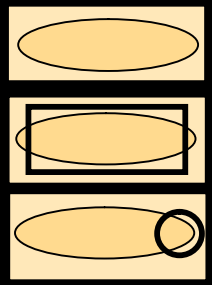
- Targeted abstract architecture for MPEG4 encoder
- Flexible and Scalable Interconnect Structure
- Imposes some architecture configurations
  - Splitter and Combiner → IP
  - Number of tasks per CPU → 1

Outline

- Introduction to High-Level Architecture Exploration
- Flexible Architecture Model for Architecture Exploration
- Flexible Algorithm and Architecture Representation
- High-Level Architecture Exploration Flow for MPEG4 Encoder
- Conclusions

Flexible Algorithm/Architecture Model for MPEG4





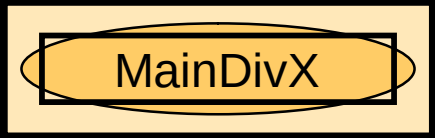
Fixed task

Task with flexible computations (belonging to a SIMD)

Task with flexible input/output (connected to a SIMD)

- Model described using macro language (i.e. Rive, M4)
- Tasks description language: C/C++
- Tasks encapsulated into SystemC modules
- Communication done using Message Passing primitives
- Communication infrastructure: MPI-SystemC High-Level Parallel Programming Model

Tasks with Flexible Computations




```
//----------------- MainDivX task ----------------------------------------
EXTERN *image_memory"N",height"N", length"N", top_border"N", left_border"N",
            bottom_border"N", right_border"N",&result"N"  ;

void MainDivX"N"_MAIN ( )
{
  //initialization of computations
  MainDivX"N"_INIT (&image_memory"N", height"N", length"N");

  //data_receive_communication from the Splitter
  MPI_"PROTOCOL"Recv(this,&image_memory"N",sizeof(image_memory"N"),
                                "DATA_WIDTH",SPLITTER_ID,22,MPI_COMM_WORLD);

  //calls the function with flexible computations
  MainDivX"N"_COMPUTE (&image_memory"N",height"N", length"N",
                                top_border"N", left_border"N",
                                bottom_border"N", right_border"N",&result"N");

   //send_results_communication to the VLC
   MPI_"PROTOCOL"Send(this,&result"N",sizeof(result"N"),"DATA_WIDTH",
                                VLC["target_vlc"]_ID,22, MPI_COMM_WORLD);
}
```

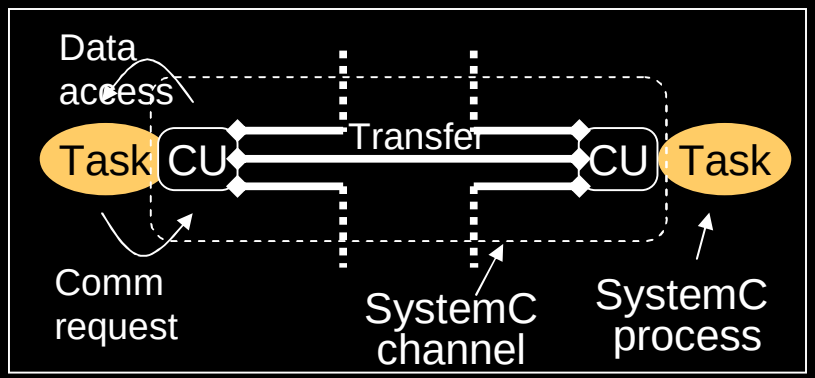
- Described using macro language
- As customizable as possible
- Represents a template description used later to generate multiple MainDivX tasks

Communication Infrastructure: MPI-SystemC HLPPM

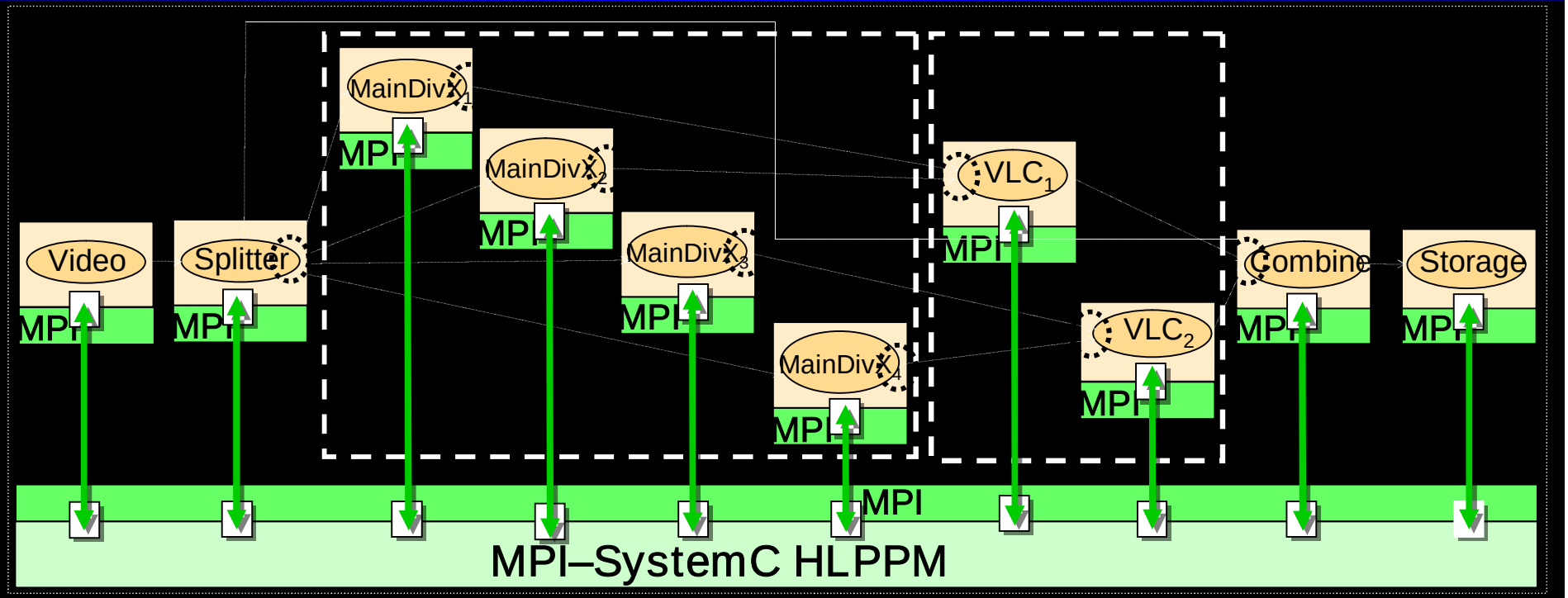
- Tasks communicate using Message Passing primitives

```
MP_Init(*this,argc,argv);
MP_Finalize(*this);

MP_[I]Send(*this,buf,count,datatype,dest,tag,comm);
MP_[I]Recv(*this,buf,count,datatype,source,tag,comm,status);

MP_[I]BSend(*this,buf,count,datatype,dest,tag,comm);
MP_[I]BRecv(*this,buf,count,datatype,source,tag,comm,status);

MP_[I]SSend(*this,buf,count,datatype,dest,tag,comm);
MP_[I]SRecv(*this,buf,count,datatype,source,tag,comm,status);

MPI_Wait(*this,request,status);
MPI_Test(*this,request,flag,status);
```

- MPI-SystemC High-Level Parallel Programming Model
  - Similar with MPICH
  - Implemented in SystemC
  - Capable of time annotating the communication
  - Splits a communication in 3 phases: initialization, data transfer, release

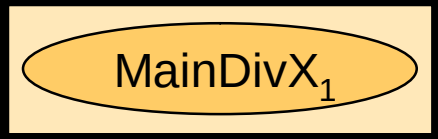


Executable SystemC Model of Combined Arch./Alg. [1/2]



- Obtained after macro-generating the Flexible Algorithm/Architecture Model
- Different Configuration Parameters → Different Executable SystemC Models
- Executable model
- Un-timed model
- Used for algorithm debugging, synchronization debug, communication debug, performance analysis

Executable SystemC Model of Combined Arch./Alg. [2/2]

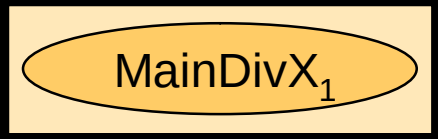


Macro-generated code for MainDivX$_1$

```
//----------------- MainDivX task ----------------------------------------
EXTERN *image_memory1,height1, length1, top_border1, left_border1,
            bottom_border1, right_border1,&result1 ;

void MainDivX1_MAIN ( )
{
  //initialization of computations
  MainDivX1_INIT (&image_memory1, height1, length1);

  //data_receive_communication from the Spliter
  MPI_Recv(this,&image_memory1,sizeof(image_memory1),
              32,SPLITTER_ID,22,MPI_COMM_WORLD);

  //calls the function with flexible computations
  MainDivX1_COMPUTE (&image_memory1,height1, length1,
                          top_border1, left_border1,
                          bottom_border1, right_border1,&result1);

   //send_result_communication to the VLC
   MPI_BSend(this,&result1,sizeof(result1),32,
                  VLC[0]_ID,22, MPI_COMM_WORLD);
}
```

Timed Executable SystemC Model [1/2]

- Obtained after time annotating the computations and communication in the Executable SystemC Model with Combined Algorithm/Architecture




```
//------------------ MainDivX task ----------------------------------------
EXTERN *image_memory1,height1, length1, top_border1, left_border1,
            bottom_border1, right_border1,&result1 ;

void MainDivX1_MAIN ( )
{
  //initialization of computations
  MainDivX1_INIT (&image_memory1, height1, length1);
  WAIT(13.224);


  //data_receive_communication from the Spliter
  MPI_Recv(this,&image_memory1,sizeof(image_memory1),
              32,SPLITTER_ID,22,MPI_COMM_WORLD);

  //calls the function with flexible computations
  MainDivX1_COMPUTE (&image_memory1,height1, length1,
                              top_border1, left_border1,
                              bottom_border1, right_border1,&result1);
  WAIT(2.312.564);

   //send_result_communication to the VLC
   MPI_BSend(this,&result1,sizeof(result1),32,
                  VLC[0]_ID,22, MPI_COMM_WORLD);
}
```

Timed Executable SystemC Model [2/2]

Computation time annotations

- Computation times depend on the type of CPU and are determined using an ISS simulation
- Can be considered fixed computation times (an average) or variable computation times read from a table file
- Annotation granularity vs. Work Effort

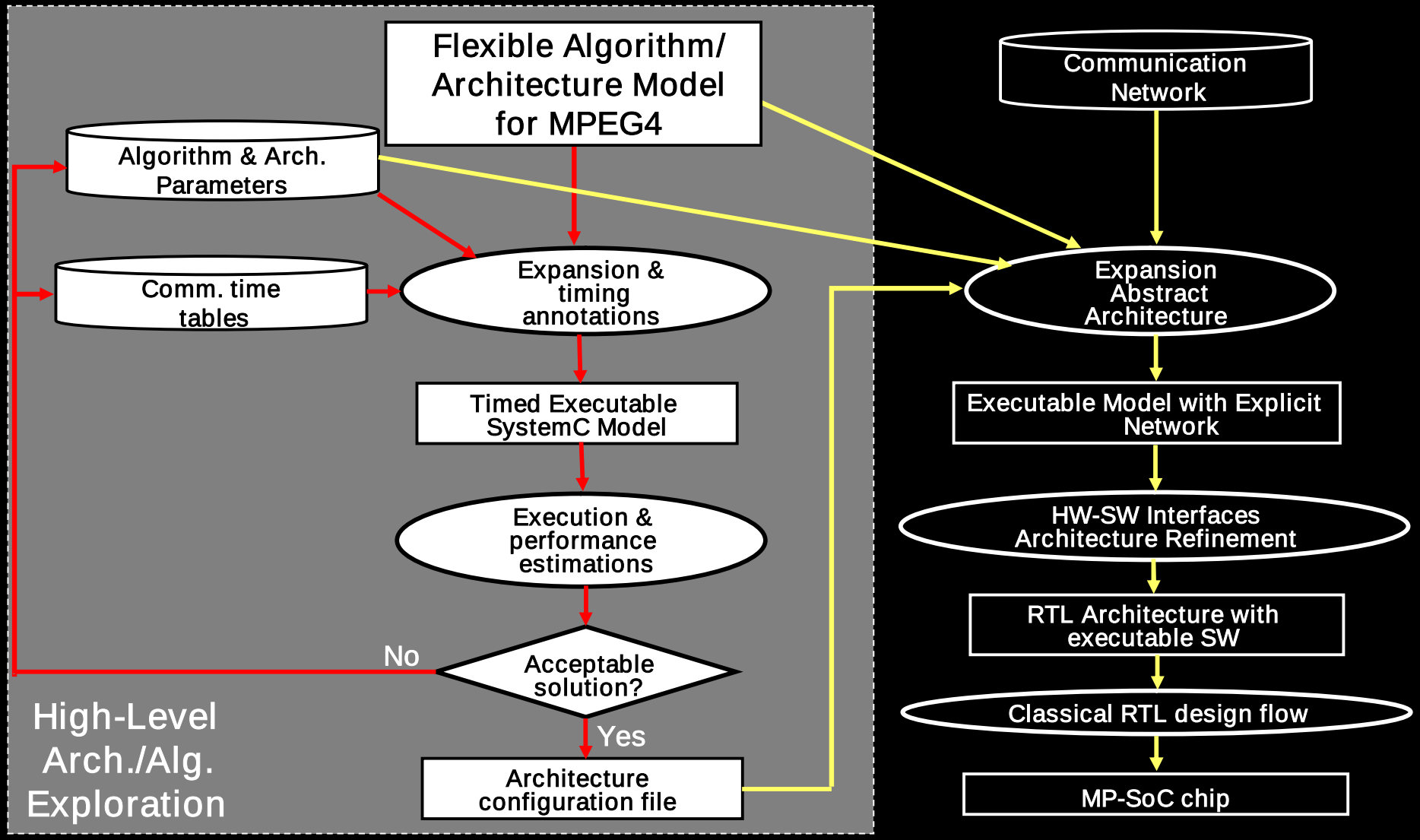
Communication time annotations

- Automatically integrated and managed by the MPI-SystemC HLPPM
- Splits every communication in 3 steps : initialization, transfer, release
- Time annotation for each step depends on the used MPI parameters

Outline

- Introduction to High-Level Architecture Exploration
- Flexible Architecture Model for Architecture Exploration
- Flexible Algorithm and Architecture Representation
- High-Level Architecture Exploration Flow for MPEG4 Encoder
- Conclusions

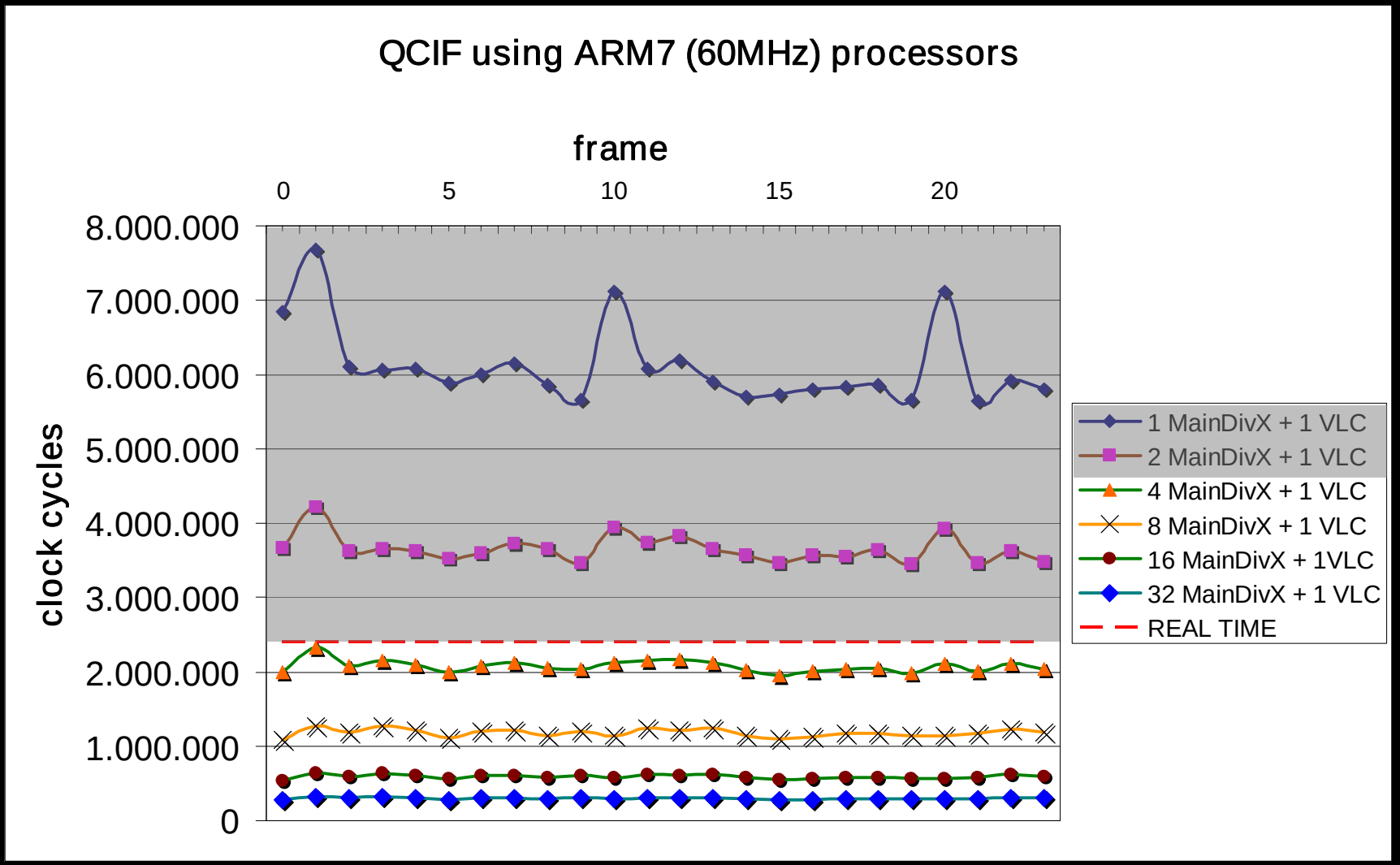
Detailed Representation of the Design Flow



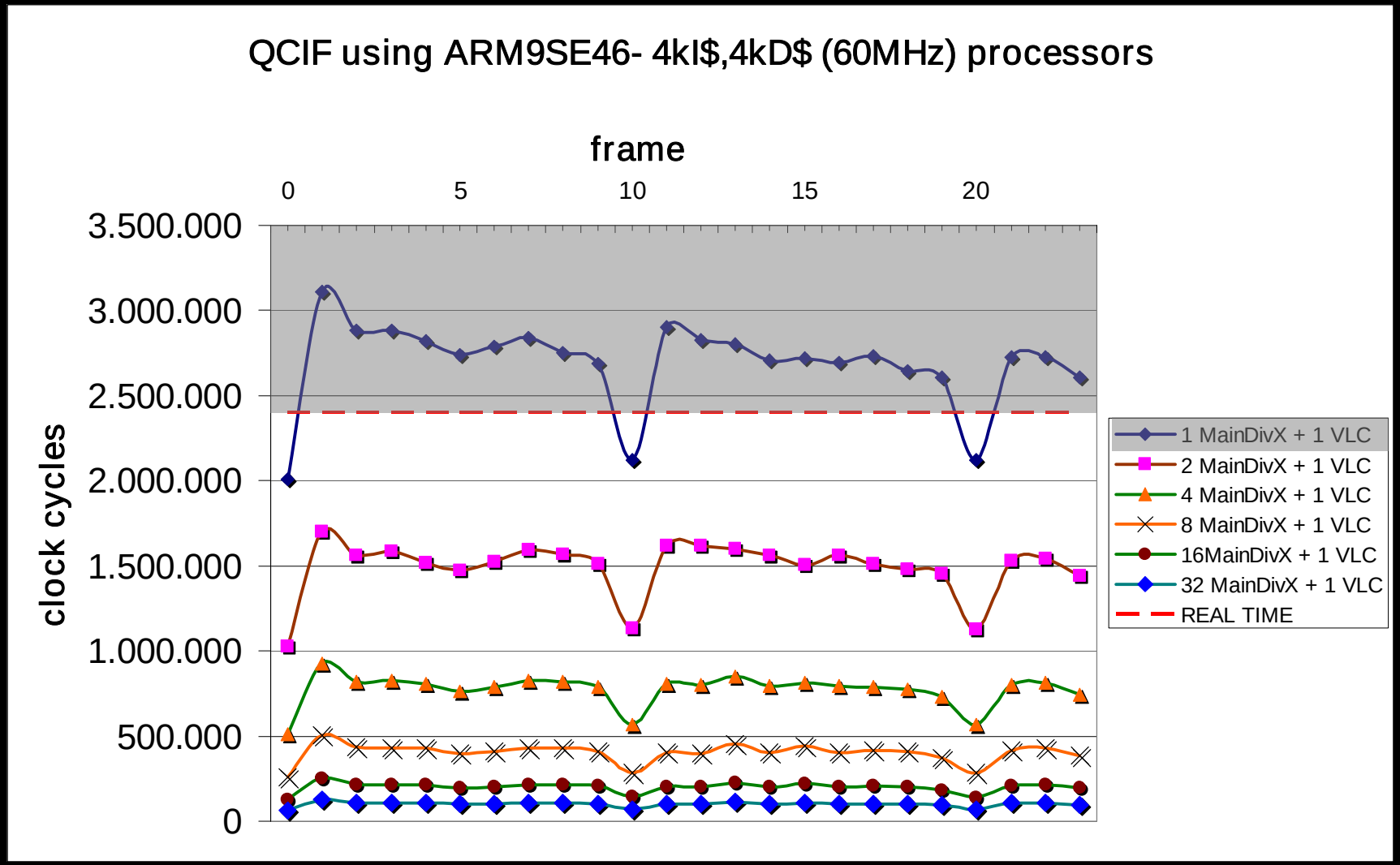
Execution and Performance Estimations

- Executable Algorithm/Architecture Model
  - generated automatically according to the chosen algorithm and architecture configurations
  - insert time annotations
- Executing the Executable Algorithm/Architecture Model
  - SystemC compilation
  - Execute
  - Performance measured using sc_simulation_time() function after encoding every frame
- Results representation
  - Using graphic table for representing the performances measured during the simulation
  - Compare the obtained performances for different algorithm/architecture configurations

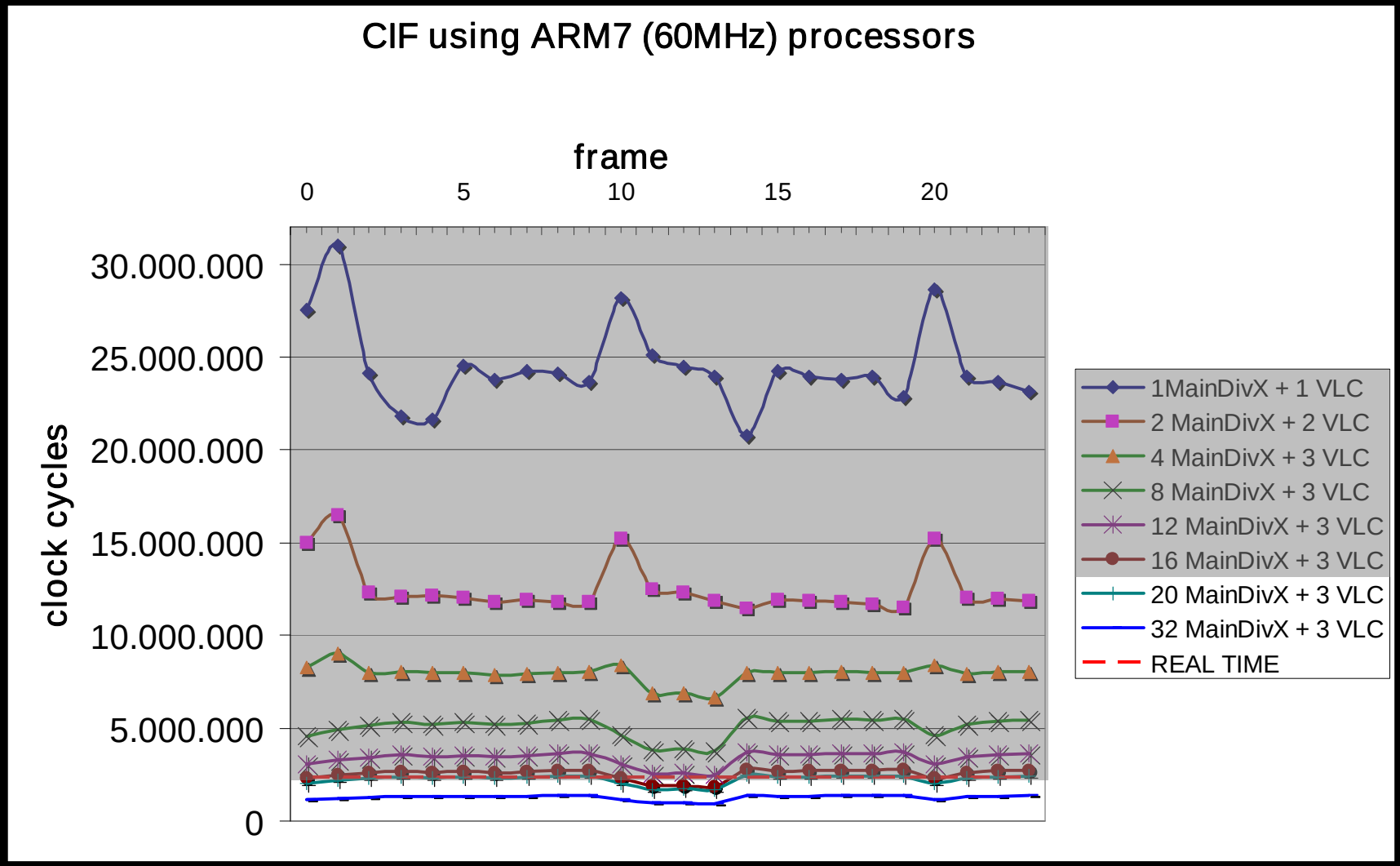
Performance Estimations: QCIF, 25fps, ARM7



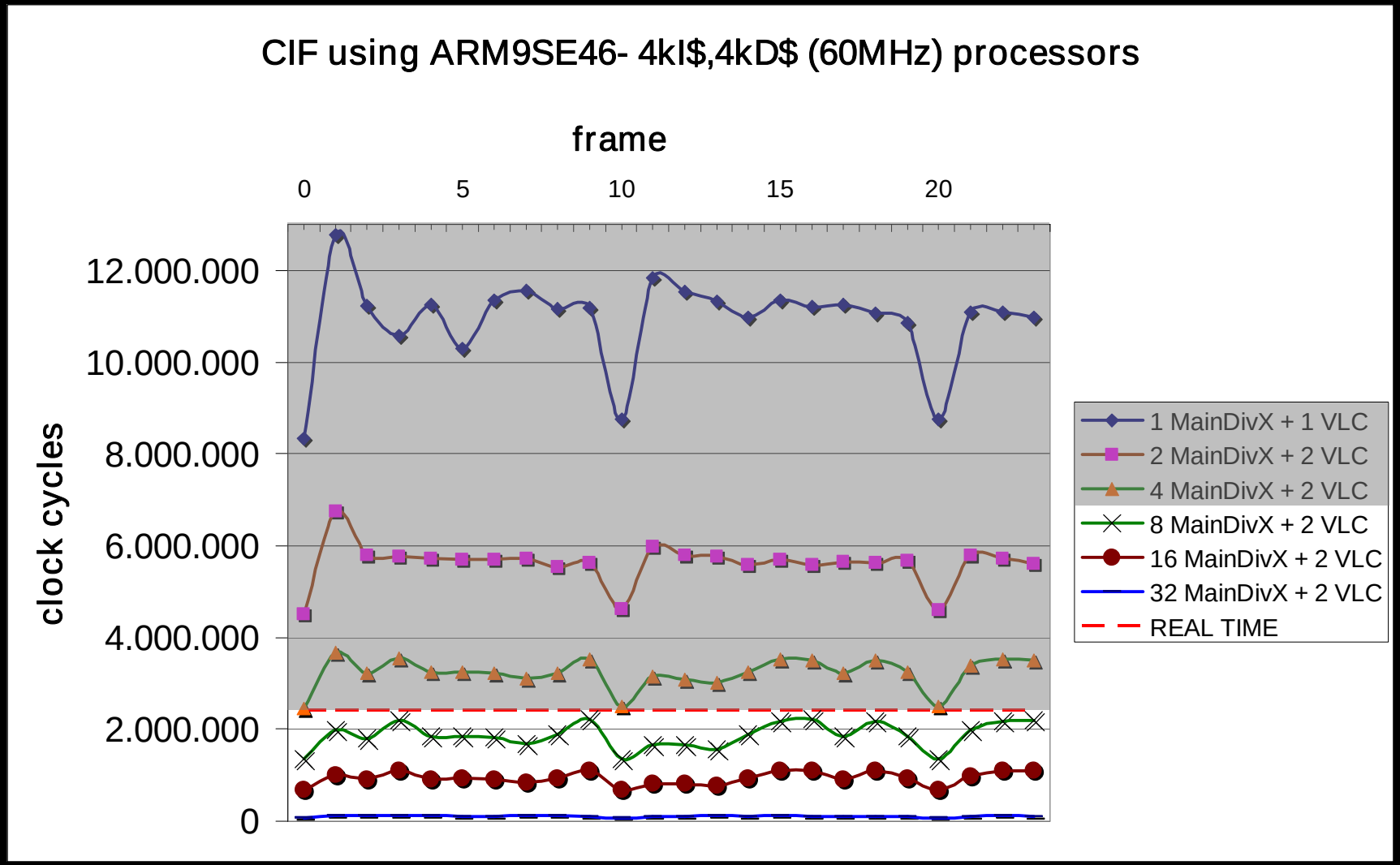
Performance Estimations: QCIF, 25fps, ARM9



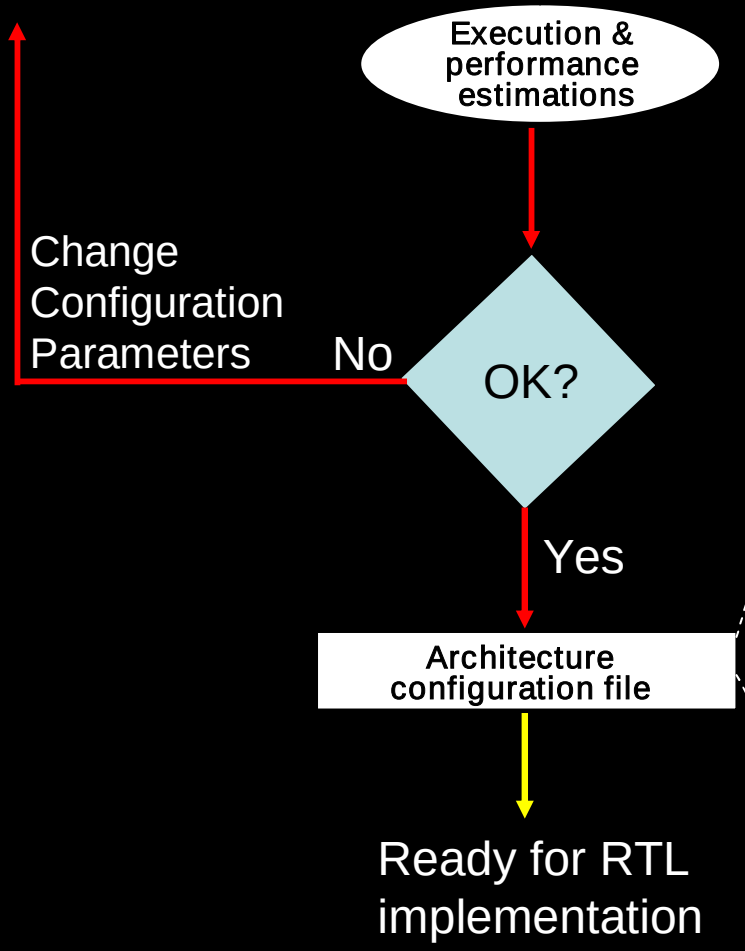
Performance Estimations: CIF, 25fps, ARM7



Performance Estimations: CIF, 25fps, ARM9



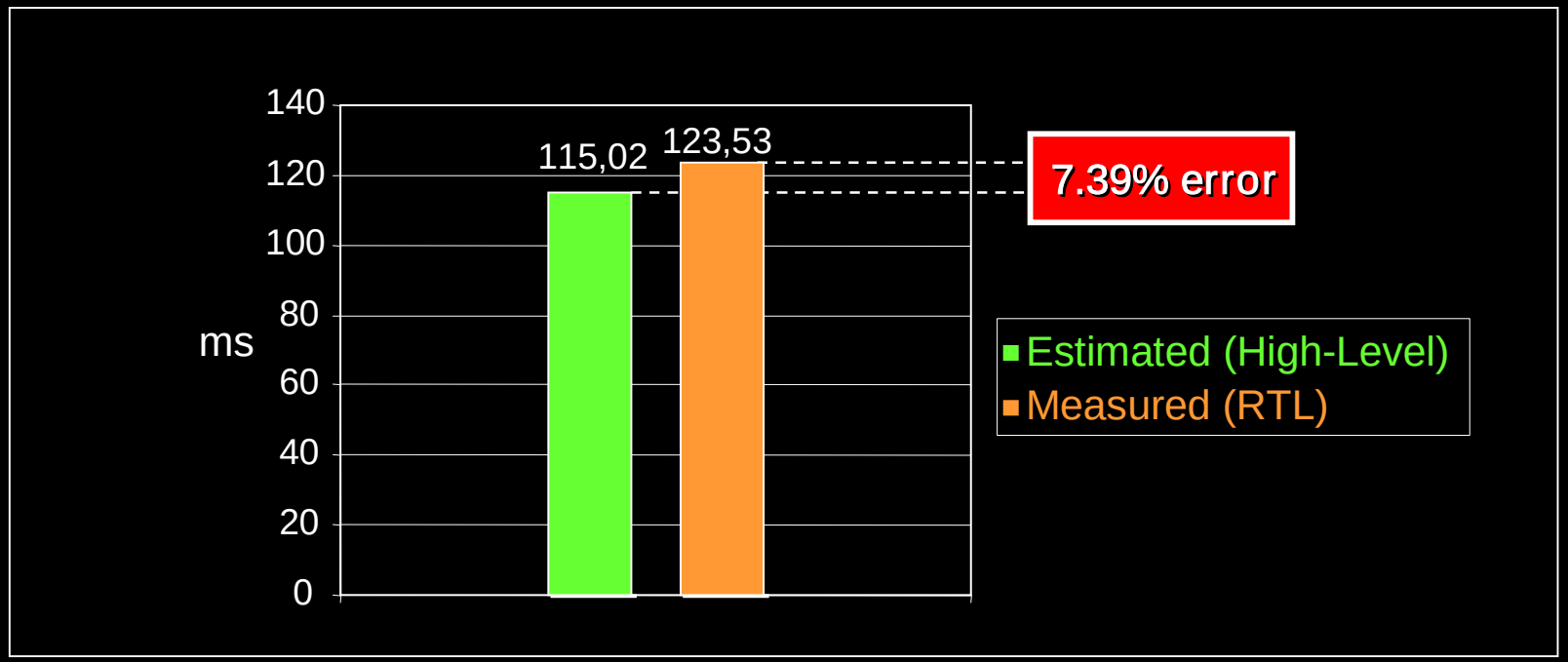
Architecture Exploration



CPUs Number → 5 (4 MainDivX + 1 VLC)
IPs Number → 2
Splitter → IP
$MainDivX_1$ → CPU (ARM7)
$MainDivX_1$ → 60MHz
$MainDivX_1$ → no cache
… the same for the other 3 MainDivX
VLC_1 → CPU (ARM7)
VLC_1 → 60MHz
VLC_1 → no cache
Combiner → IP
Splitter-$MainDivX_1$ send protocol → Blocking
$MainDivX_1$-Splitter recv. protocol → Non-Blocking
Splitter-$MainDivX_1$ burst size → 128 bytes
Splitter-$MainDivX_1$ data_width → 32 bits
Splitter-$MainDivX_1$ init_latency → 2 cycles
Splitter-$MainDivX_1$ data_latency → 3 cycles
$MainDivX_1$-VLC_1 send protocol → Blocking
VLC_1-$MainDivX_1$ recv. protocol → Blocking
VLC_1 recv. arbitration → AnySource
$MainDivX_1$-VLC_1 burst size → 810 bytes
$MainDivX_1$-VLC_1 data_width → 32 bits
.... similar for the other modules

- Currently, all the decisions are done manually by the designer

High-Level Estimation vs RTL Measurements



- High-Level Estimation = sufficient precision to assure its reliability
- Error caused by the impossibility to capture at High-Level:
  - OS (scheduling, service calls, latencies induced by the API calls)
  - Interconnect between the CPU buses and Network Interfaces → GRANT on bus
  - HW/SW Wrappers

Results Analysis

- Proposed exploration at High-Level (i.e. for QCIF using ARM7)
  - ~15 minutes were required to generate the Timed SystemC Model
  - ~2 minutes were required to simulate 25 frames
  - ~1 hour was required to find one architecture solution
- Exploration at RTL level for QCIF using ARM7
  - ~6 months were required to build manually the RTL architecture
  - ~25 hours were required to simulate 25 frames
  - testing different architecture configuration at RTL level becomes unacceptable
- Adaptation to other applications
  - Flexible Algorithm/Architecture Model has to be remade.
  - MPI-SystemC has to be readapted only in case new communication topologies are required

Outline

- Introduction to High-Level Architecture Exploration
- Flexible Architecture Model for Architecture Exploration
- Flexible Algorithm and Architecture Representation
- High-Level Architecture Exploration Flow for MPEG4 Encoder
- Conclusions

Conclusions

- MPEG4 video encoder requires complex and long design time MP-SoC implementations
- High-Level architecture exploration helps finding reliable architecture configurations before the MP-SoC implementation starts
- A unique Flexible Algorithm/Architecture Model for MPEG4 Encoder were used to automatically generate different Algorithm/Architecture Models required during the exploration
- Time annotations were used to simulate the performances of computations and communications running together
- Multiple configurations were experimented
- The High-Level performance estimations are close to the RTL performance measurements
- The proposed approach drastically reduced the time to explore different configurations of MPEG4 Encoder in MP-SoC
- This method can be extended to other applications

Thank you

MARIUS BONACIU

TIMA Laboratory, SLS Group

46 Avenue Félix Viallet, Grenoble, FRANCE

Tel : +33 4 76 57 43 34

Fax : +33 4 76 47 38 14

marius.bonaciu@imag.fr