Rapid application development on multiprocessor reconfigurable systems

Linfeng Ye, Jean-Philippe Diguet, Guy Gogniat

Lab-STICC, team MOCS
CNRS - European University of Brittany,
Lorient-Brest, France
Agenda

1. Context
2. XPSoC Proposed approach
3. API
4. Decision
5. Case study
6. Conclusions
FPGA for Embedded System Design

Theoretical Pros

- Ability to design multiprocessor systems on a single FPGA
- Improve weak soft-core performances by specializing coprocessors/accelerators according application needs
- Dynamic-Partial reconfiguration available to adapt architecture to application requirements at runtime

Real Cons

- Productivity and Design Costs remain the main issue
- Error-prone, Time Consuming CAD tools and methodologies
- Dynamic-Partial reconfiguration today:
  ⇒ Prohibitive design and programing efforts
  ⇒ No straightforward programming models
Observations

- Embedded systems: intensive reuse of standard functions
  (Signal processing, Image, Video, Network, Security)

- Success of TI in embedded systems based on fixed heterogeneous architectures (Proc, DSP, Coprocessors) + API

- Taking advantage of reconfiguration capabilities:
  - Reuse strategy based of architecture models
  - Maintenance / Upgrade of long-life products
    (e.g. network routers, satellite systems)
  - Save hardware Context / Data dependent applications
    (e.g. security protocols, software defined radio, mobile smart cam)
Global Approach: Models and API

1. Architecture Models
   - Architecture virtualization
   - MDE methodology and tools (code and script generation)

2. Programing Model based on Hardware independent API for standard function fast implementation

3. Configuration Decision (binaries, bitstreams)
   - 2 steps (Offline + Online)
   - Embedded decision
   - Hierarchical Bitstream Repository
Global Approach

MPSoC Meta-Model (e.g. OMG UML MARTE + New Features for Reconfiguration)

Model Transformations Based on Rule composition and P,H,S Meta-models

Platform Model

Hardware Model

Software Model

Code Generation (e.g. Xilinx + Home tools)

EDK Project

HLS Scripts
e.g. Copro: GAUT, Madeo
NoC: μspider II

API-based Embedded SW
e.g. Xtasks + Decision / Configuration Manager

Constraints:
Latency + I/O FSL

Configuration Servers

STD Functions

XPSoC API

Lab-STICC

J-Ph. Diguet, Lab-STICC
FPL’10, Milano
Agenda

1. Context

2. XPSoC Proposed approach

3. API

4. Decision

5. Case study

6. Conclusions
XPSoC summary

- **Scalable and reconfigurable architecture model** for data-flow applications and based on Xilinx/Micro-Blaze.
  - 1 Master (**XManager**) : Multi-tasks (Peta-Linux)
  - N Slaves (**XWorker**) : Mono-task (µKernel : Memory Access)
  - Reconfigurable components (**XModules**) connected to FIFO or common Bus (**XAccelerators**)

- Communications: **Message Passing and Shared memory**
- Programming: **XTask and XFunc APIs**.
- **XModules Library** (for reconfigurable IPs management)
- Configuration Cache (local / server)
- Performance-oriented **self-reconfiguration decision-making**
XPSoC software architecture
Design flow

1- Application analysis: Profiling
Identification of critical functions for XTask implementation

2- Selection of a XPSoC model from a library of models

3- Application Development / Migration based on XTask APIs

4- HW and SW Implementations based on XFunc APIs

5- XModules implementation from design reuse or development with HLS tools according to standard XFunc

6- Generation of binaries and partial configuration bitstreams if unavailable from servers

7- Test and Validation

J-Ph. Diguet, Lab-STICC

FPL’10, Milano
Agenda

1. Context
2. XPSoC Proposed approach
3. API
4. Decision
5. Case study
6. Conclusions
Objectives

**XTask API**
- Create, suspend, kill XTasks (reconfigurable HW/SW Tasks)
- Synchronization between XManager and XWorkers
- Management of XModules (reconfigurable resources)
- On-the-fly reconfiguration decision
- Control of dynamic partial reconfiguration

**XFunc API**
- Provide a unified programming prototype for both HW and SW versions
- A set of generic functions that can be specialized to handle various application domains
**XTask API**

- **XTaskReadFunctionList** gets reconfigurable function list.
- **XTaskXModuleInit** initializes XPSoC with a profile-based solution.
- **XTaskCreate** creates a XTask for *FUID* function in main memory.
- **XTaskRunNonBlocking** copies XTask to shared memory and launches a non blocking task (resp. blocking task).
- **XTaskUpdateFunctionList** updates the function list by order of priority.
- **XTaskUpdateSystem** updates the XPSoC architecture by running dynamic partial reconfiguration if necessary.
Application development and migration: XTaskAPI

```c
int main (argc, argv)
{
    ...
    XTask_t *xt=null;
    XDesc_t *xdesc=null;
    XFunction_t *fl = XTaskReadFunctionList (funclist);
    XPARosC *xp = XPARosC_ModuleInit (conflist);
    switch (operation)
    {
    case D:
        /* Decryption */
        fuid = IMAGE_PGM_D;
        sol = XTaskFindSolution (fuid, in->width*in->height);
        if (sol > 0)
        {
            XTaskDelete (xt);
            XTaskUpdateFunctionList (fuid, x_size*y_size, xtime);
            if (sol > 0)
            {
                xt = XTTaskCreate (sol, fuid, xdesc, inAddr, inSize, outAddr);
            }
        }
        else {
            xtime = XTaskImageProcessing (IMAGE_PGM_D, inAddr, inSize, outAddr, xdesc);
        }
        XTaskUpdateSystem (xt);
        break;
    } /* */
    case E:
    }
    ...
}
```

```c
XTask_t *XTaskCreate (unsigned int xwid, unsigned int fuid
, XDesc_t *XDesc, unsigned int inAddr
, unsigned int inSize, unsigned int outAddr);
{
    ...
    xt = (XTask_t *)malloc (sizeof (XTask_t));
    xt->xtaskID = xtaskCounter++;
    xt->xbaseAddr = (xReg[xwid>>SHIFT_XWID] + (xwid&MASK_IFID));
    xt->status = status;
    xt->xfuction = fuid;
    xt->xdescription = xdesc;
    xt->inAddr = inAddr;
    xt->inSize = inSize;
    xt->outAddr = outAddr;
    return xt;
}
```

```c
int XTskRunBlocking (XTask_t *xt)
{
    unsigned int xbaseAddr = xt->xbaseAddr;
    XTskWriteReg (xbaseAddr, STATUS_OFFSET, xt->status);
    XTskWriteReg (xbaseAddr, FUNC_OFFSET, xt->xfuction);
    XTskWriteDesc (xbaseAddr, xt->xdescription);
    XTskWriteReg (xbaseAddr, INPUT_ADDR, xt->inAddr);
    XTskWriteReg (xbaseAddr, INPUT_SIZE, xt->inSize);
    XTskWriteReg (xbaseAddr, OUTPUT_ADDR, xt->outAddr);
    XTskWriteReg (xbaseAddr, CONTROL_OFFSET, XTASK_RUN);
    while (XTskReadReg (xbaseAddr, CONTROL_OFFSET) !=
        XTASK_STOP)
        usleep (100);
    return 0;
}
```
XFunc API / per application domain

- XFuncSpeechDecoder
- XFuncSpeechEncoder
- XFuncAudioDecoder
- XFuncAudioEncoder
- XFuncImageDecoder
- XFuncImageEncoder
- XFuncImageProcessing
- XFuncDecrypt
- XFuncEncrypt
Application development and migration: XFuncAPI

void D (gmap *in, gmap *out, unsigned int key)
{
    unsigned int data, bit, i, res, size;
    unsigned int *temp, *datain;
    temp=(unsigned int*)(out->raster);
    datain=(unsigned int*)(in->raster);
    size=(in->width*in->height)/4;
    for (i=0; i<size; i++)
    {
        data=*(datain+i);
        for (bit=0; bit<32; bit++)
        {
            if (bit%2 == 0) {
                res |= ((((data >> bit) ^ (key >> bit)) & 0x1UL) << bit);
            } else {
                res |= (((~((data >> bit) ^ (key >> bit))) & 0x1UL) << bit);
            }
        }
        *(temp+i)=res;
        res=0;
    }

    key = XTaskGetNextDesc (desc);
    void D (in, out, key){ ... }

Agenda

1. Context
2. XPSoC Proposed approach
3. API
4. Decision
5. Case study
6. Conclusions
Key parameters for run-time configuration decision-making

**Self-adaptivity tuning**

- Processing / Self-adaptivity task parallelism: blocking and non-blocking XTask.
- Granularity of XTask: Instruction, Configuration, Data.
- Frequencies of reconfiguration, observation and decision tasks.
- Reactivity/Reconfiguration overhead tradeoff (smoothing factor).
Decision-making

Algorithm

- Get critical function list

- Xmodule initialization based on off-line profiling

- For all critical functions repeat:
  1. Get observation data (execution times, #data)
  2. Update function speed-up metrics (per data and application)
  3. Fast sorting (N first candidates)
  4. Configuration decision including configuration cost.
On-the-fly reconfiguration decision

input_F2 = buffer_size

input_F2' = buffer_size / 10

C2
XW1
C1
XM

Execution Time

Decision Time
Reconfiguration Time
HW XTask Execution Time
SW XTask Execution Time
Communication Time
Agenda

1. Context
2. XPSoC Proposed approach
3. API
4. Decision
5. Case study
6. Conclusions
Implementation on FPGA

- XManager (Microblaze)
- XWorker (Microblaze)
- Coprocessor-1
- Coprocessor-2
- UART
- Ethernet
- SDRAM
- Shared Memory
- Mutex
- HWICAP

- OPB
- Reconfigurable part

J-Ph. Diguet, Lab-STICC
FPL'10, Milano
Image decoding example with data dependent parameters
On-the-fly Reconfiguration and Data Granularity Impact

![Graph showing execution time vs. MAX XTask Granularity](image)

Variable granularity
On-the-fly reconfiguration scenario

Initial Configuration

Reconfiguration (X, Y, Z) : F3 -> F2

Parameter Modification: F2 Input=100B

Reconfiguration (Y) : F2 -> F3

Reconfiguration (Z) : F2 -> F3

End of execution (X)

End of execution (Y)

End of execution (Z)
Agenda

1. Context
2. XPSoC Proposed approach
3. API
4. Decision
5. Case study
6. Conclusions
Conclusions

Conclusion

- Development and test of a design flow to speed-up reconfigurable architecture design:
  - Model-based multiprocessor reconfigurable architectures
  - API for rapid development based on standard function libraries
  - On-line reconfiguration decision-making
- Self-adaptivity: no future without high-level programing models

On going / Perspectives

- Software code generation
- Reconfigurable IP library (Interface HLS / FSL Ok)
- Cache strategy for configuration files (remote server/local memory)
- Collaborations are welcome...