A Process Model suitable for defining and programming MpSoCs

MpSoC-Workshop at Rheinfels, 29-30.6.2010

F. Mayer-Lindenberg, TU Hamburg-Harburg

1. Motivation
2. The Process Model
3. Mapping to MpSoC
4. The Component Library
5. Implementation Remarks

Ref.: High-Level FPGA Programming through Mapping Process Networks to FPGA Resources, ReConfig '09, Cancun, 2009

Remark. This work is not on mapping applications to hardwired single-chip processor networks. It does map applications to single-chip networks but to such compiled to exactly fit the application with a trivial 1-1 mapping step. The method also holds for ASIC design. The target FPGA can host some limited number of processors. The work starts from a process model not supported by C.

Motivation

1) Efficient FPGA usage heavily depends on sequential control of sub circuits. A complex application specific circuit can be used multiple times within the processing delay allowed by the application. A complex compute circuit plus a sequential controller is similar to a CPU sub system. Heterogeneous MpS on FPGA is therefore considered to be a natural approach to FPGA usage.

2) The definition of the network of processors to be mapped onto the FPGA is considered to be an intermediate design step needed to exploit the FPGA resources. It should be automated and be performed by a compiler transforming a high-level description of the application processing to a conforming FPGA realization.

3) The application model must specify the desired processing with enough detail to be able to perform the compilation. It has to show the parallelism that can be exploited by the parallel target architecture, an the timing constraints. The compilation can be supported by virtual HW-OS functions such as standard control and networking circuits.

Process model

I. Basic ingredients

- A process is an automaton that receives stream input and generates output streams; it cyclically processes all elements of the input stream. The inputs synchronize the processing; the outputs don't. Input frequencies are specified, and, for the outputs, time steps from the start of the processing cycle that also limit the processing time available to generate them.

- An application system is descibed as a network of processes communicating with each other or with external processes. Streams 'are' pairs of processes.

Discussion:

A process may be hierarchically composed of communicating sub processes which permits the integration of the application modeling into a programming language. The process model has been implemented in an experimental language 'π-Nets' and in a multi-threaded C dialect.

So far, the process model is similar to a timed version of a dataflow model like Kahn's. An application can have many equivalent models. The canonical choices are the maximally decomposed model and one in which the internal processes don't communicate with each other.

Processes may have private memory. Each variable is written to by a unique process. There are no sub automata shared by several processes unless they are purely functional. The application processes are defined statically.

Process model cont'd

II. State sampling

Our process model uses a secondary non-dataflow network for letting the processes communicate, namely by allowing to asynchronously input from the state of other internal or external processes and, vice versa, to generate visible state data for other processes. Time conditions depending on the state data read from other processes may be defined and waited for. The output of visible state is subject to the time steps of the process. A system model with no stream communications between internal processes will still use state sampling to let them cooperate. State sampling is under the time control of the reading process.

III. Process groups

Another useful concept, although not viable for the intended mapping to MpSoC, is the inclusion of process groups sharing a top level control flow. Processes in a group also synchronize at the start of every execution cycle. They may include sub processes in which some group members participate; these then synchronize on entry into the sub process. Process groups generally describe collective work on a task with some common control.

Remarks. Process groups can be used to implement dynamic resource management. Examples are the reconfiguration of HW resources performed by a group member between subsequent sub processes, or dynamically activating a process using extra resources. They also serve to define parallelism within a composite process, also as a starting point for optimizations. So far, the process grouping is supported by π-Nets and its implementation only.

Features within the definition of a process group (π-Nets)

```
1    F64                                    .. selecting the data type
2
3    CPC x name                             .. process header, stream x triggers
4    {
5    #1                                     .. member process 1 starts
6         x + 1 → h                         .. instructions for member process 1
7
8    #2                                     .. member process 2
9         $$ 1.0                            .. continue at 1.0 second from start
10        2*h >> y                          ..  implicit transfer of h
11                                          .. instructions for member process 2
12
13        if  h > 4                         .. affects processes 1 and 2
14
15             {                            .. sub process, 1 and 2 synchronize
16                  h >> y1
17             #1                           .. process 1 part of sub process
18                  if h > 5                .. affects process 1 only
19
20                       h >> x1
21             }                            .. no 'else' part in this example
22
23             stop                         .. no further process cycles
24   }
```

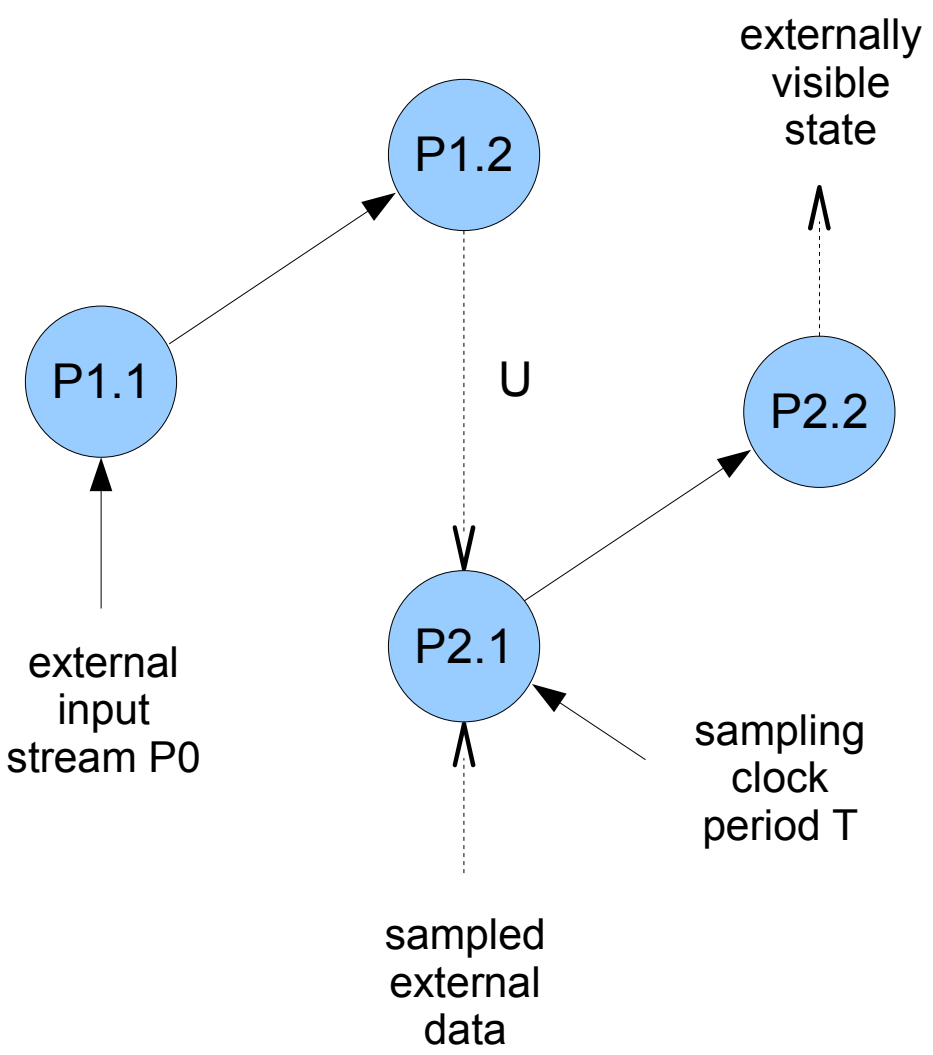
Mapping an application to an MpSoC

Mapping is based on a library of predefined component types, a standard instruction memory based controller, several ALU circuits attachable to the controller, point-to-point unidirectional link interfaces, and memory controllers.

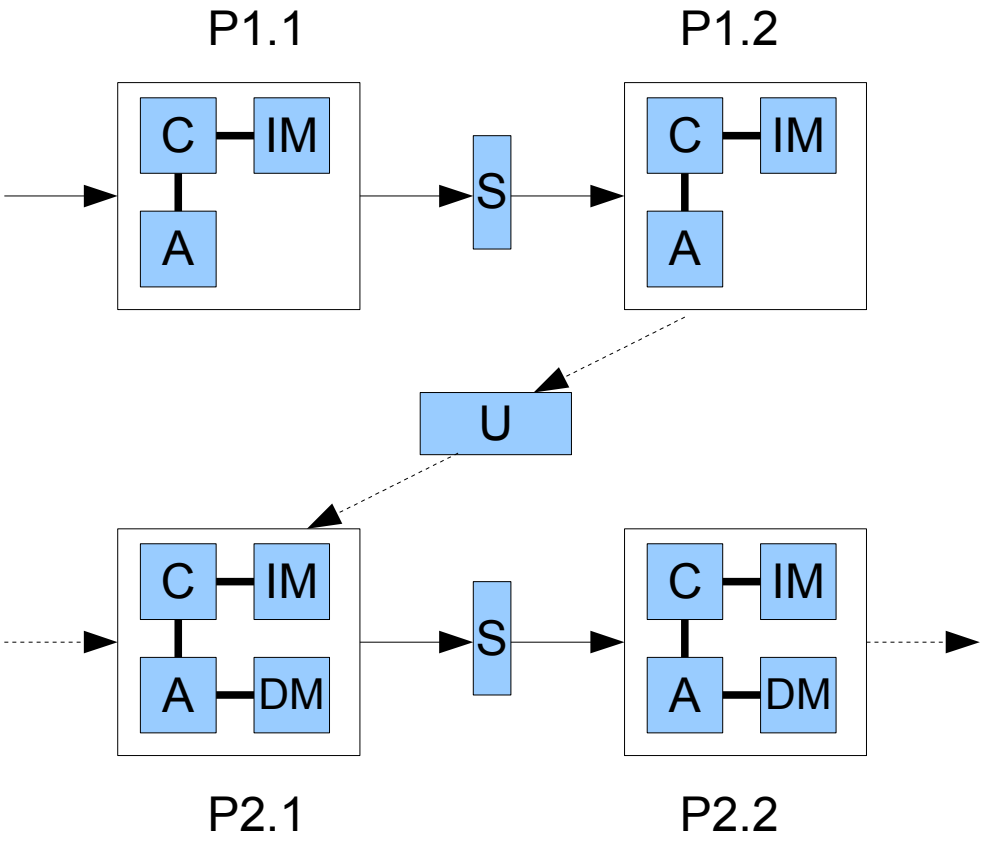
- for every individual process or member of a process group a controller is instanciated with one to several blocks of IRAM, and an ALU circuit for the data type(s) used in that process; the ALU gets extra RAM if needed
- if two processes perform stream communication with one another, synchronous link interfaces are allocated and attached to the CPUs
- if two processes communicate by one reading a state variable of the other, a dual ported RAM is allocated between them and wired up accordingly; for large data structure this will be a block RAM, otherwise distributed RAM
- external stream ports and asynchronous ports are available in the library and mapped accordingly

The processor network duplicates the process network as defined by the application model. It does not rely on a predefined infrastructure but is compiled as an application specific network to be realised through the FPGA routing. After performing the mapping, an estimate of the required FPGA resources becomes available immediately, and the compiler can proceed to code generation for the individual controllers with their standard ISA and to provide performance and efficiency data.

An example



Measurement system with UI



Corresponding MpSoC structure

Remarks on the component library

The ALU circuit controller is a low complexity design adapted to the FPGA resources (18-bit instructions, register banks in distributed RAM etc.). Besides generating the instruction sequence for the ALU, it provides the control flow, memory control and communications.

- half of the 2^18 instructions are ALU instruction output as a 17-bit word
- ALU may use data and a memory of its own having a different word size
- data memory supported by instructions and control signals
- dual threads to support pipelined ALU circuits (running at different priorities)
- DMA channel for parallel i/o transfers and accesses to external memory
  DMA address usable as an extra address pointer for stream ops otherwise
- i/o and DMA transfers both for the controller and the ALU
- automatic synchronization and context switches for stream i/o

The link interface is 18-bit wide and provides a small FIFO realized with distributed RAM and handshaking. The CPU does not provide an external memory bus but just i/o ports for attached link interfaces and other handshaking interfaces.

Some hints on the implementation

- Intermediate code represents processes as separate CDFGs, identifies application processes by linking them into a list (all are defined statically)

- all processes have an attribute indicating the executing processor; processor Codes are generated in multiple selective runs through the same interm. code

- compiler generates link information along with intermediate code; can also be obtained by searching in the intermediate code for send, receive, read tokens

- a single CDFG is generated for all processes of a group, only indicating for every basic block to which process of the group it belongs. Stream communi-cations within the group are implicit and only shown as data and control token References; they need to be synthesized during code generation. This allows for easy changes to the number and distribution of sub processes.

- the intermediate code is executable on a virtual machine. This allows for a simulation of the MpSoC including its timing. The simulation can include external processes. The virtual machine can also take part in executing the Application if it includes processes mapped to the PC.

- Netlist output is as a text file transformed into structural VHDL by an extra tool.

An MpSoC used for µ-processor education