Component reuse methodology for multi-clock Data-Flow parallel embedded Systems

Anne Marie Chana, Patrice Quinton, Steven Derrien

Component reuse methodology for multi-clock Data-Flow parallel embedded Systems

Anne Marie Chana* — Patrice Quinton** — Steven Derrien***

* National Advance School of Engineering, University of Yaounde I, Cameroon
anne_chana@yahoo.fr
** ENS Cachan Bretagne, Campus de Ker Lann, 35170 Bruz, France
Université européenne de Bretagne, France
Patrice.Quinton@bretagne.ens-cachan.fr
*** Université de Rennes 1, Campus de Beaulieu, 35042 Rennes cedex, France
Université européenne de Bretagne, France
Steven.Derrien@irisa.fr

ABSTRACT. The growing complexity of new chips and the time-to-market constraints require fundamental changes in the way systems are designed. Systems on Chip (SoC) based on reused components have become an absolute necessity to embedded systems companies that want to remain competitive. However, the design of a SoC is extremely complex because it encompasses a range of difficult problems in hardware and software design. This paper focuses on the design of parallel and multi-frequency applications using flexible components. Flexible parallel components are assembled using a scheduling method which combines the synchronous data-flow principle of balance equations and the polyhedral scheduling technique. Our approach allows a flexible component to be modelled and a full system to be assembled and synthesized with automatically generated wrappers. The work presented here is an extension of previous work. We illustrate our method on a simplified WCDMA system. We discuss the relationship of this approach with multi-clock architecture, latency-insensitive design, multidimensional data-flow systems and stream programming.

RÉSUMÉ. La complexité croissante des nouvelles puces et les contraintes de mise sur le marché exigent des changements fondamentaux dans la démarche de conception des systèmes. Les systèmes sur puce (SoC) basés sur les composants réutilisables sont devenus une nécessité absolue pour les entreprises de systèmes intégrés pour rester compétitives. Cependant, la conception d'un SoC est extrêmement complexe car elle englobe une série de problèmes difficiles du domaine de la conception matérielle/logicielle. Cet article présente une approche de réutilisation de composant pour la conception d'applications parallèles et multi-fréquences. Les composants flexibles sont assemblés à l'aide d'une méthode d'ordonnancement qui combine les principes des équations d'équilibre du modèle flot de données et la technique d'ordonnancement du modèle polyédrique. Notre approche permet de modéliser les composants flexibles, d'assembler et de synthétiser un système complet avec des interfaces (ou Wrapper) générés automatiquement. Le travail présenté ici est une extension des travaux antérieurs, nous illustrons notre méthode sur un modèle simplifié du WCDMA. Nous discutons aussi dans cet article de la relation entre cette approche et celles des architectures multi-horloge, des systèmes insensibles à la latence, du modèle flot de données multidimensionnels et de la programmation par flux.

KEYWORDS : flexible component, SoC, polyhedral model, data-flow model, parallelism, multi-clock architecture

MOTS-CLÉS : réutilisation de composants flexibles, SoC, modèle flot de données, modèle polyédrique, parallélisme, systèmes multi-fréquences

Received, December 12, 2012
Revised, February 18, 2014
Accepted, March 17, 2014

ARIMA Journal, vol. 18 (2014), pp. 67-92.

1. Introduction

Over the past 20 years, embedded systems have become the basis of the most advanced hardware and software technologies. During this period, the implementation of embedded systems has evolved from microcontroller to fully integrated systems-on-chip (SoC). SoCs and related technologies are driving embedded systems today and seem likely to do so in the foreseeable future. New demanding applications in terms of processing power appeared over the past 20 years, driven by PC boom, especially Multicore PC, Internet and wireless environments. These applications are found both in scientific computing and signal processing (telecommunications, multimedia processing, etc.) and more generally in cyber-physical systems [24, 25]. Processing such applications requires computing high capacity data that can be achieved only through parallel and distributed computing. Therefore, difficulties to develop those applications are primarily due to the exploitation of data and arithmetic parallelism, time and resource constraints.

Usually such systems are designed by re-using existing software or hardware modules often called *flexible blocks*. These flexible components are available as software or hardware module and represent specific application blocks for signal processing (DCT, FFT, etc.), telecommunications (Viterbi codes, Turbo-codes, etc.), or Multimedia (MPEG2, MPEG4, JPEG, etc.)[1]. In practice, it is not easy to assemble and operate components from different designers. Even if they are tested beforehand, there is no guarantee that, when put together the system will work correctly[32]. Integration of the components must take into account several aspects first of all, the components must be synchronized to allow a correct overall operation of the system, and to ensure proper data exchange and valid communication protocols. On the other hand, input and outputs must sometimes be buffered in order to meet synchronization constraints.

There are several Electronic System-Level (ESL) design approaches [18] based on IP reuse that aim to reduce the impact of data exchange between components or power consumption.We present in our paper a systematic method to automatically generate a hardware architecture for multi-clock, parallel data-flow systems with flexible generic components. Our design flow follows the top-down approach of ESL synthesis tools [18]. The method is based on Latency Insensitive Systems (LIS) theory and the synthesis process is preceded by the synchronous Data-flow (SDF) system description. Our scheduling technique combines the method used in the *synchronous data-flow model* [23, 30] and the *polyhedral model* method, as described in [11]. This allows to deduce the different logic sub-clocks that meet the I/O constraints of each component, and the number of additional registers that may be needed between some of the components. We also present in this paper the implementation of our approach as an extension of the high-level synthesis environment MMALPHA. MMALPHA is an academic high level synthesis tools as Spark[21], Compaan/Laura[41], GAUT[37] and UGH[2]. Note that Compaan and MMALPHA have focused on the efficient compilation of loops, and they use the polyhedral model to perform loop analysis and/or transformations.

The paper is organized as follows: in section 2 we present some related work; in section 3, we introduce the design flow of our integration approach and the system specification using ALPHA language [29]; in section 4, we present the wrapper architecture, the components assembling methodology and our model of time. In section 5 we present an

implementation of WCDMA using our approach and in section 6 we compare our approach with some related work. Finally section 7 is devoted to the conclusion.

2. Related Work

In this section, we present some works related to (1) synthesis under constraints, (2) multidimensional synchronous dataflow graph, (3) latency insensitive systems, (4) multi-clock architectures and (5) stream programming.

2.1. Synthesis under constraints

As we said in the introduction, there are several other ESL Design approaches based on IP reuse. Chavet et al [13] for example provide a design methodology to automatically generate an adapter named Space-Time Adapter (*STAR*). This approach relies on the formal modeling of communication constraints based on a Resource Compatibility Graph (*RCG*) describing timing relations between data. The architecture synthesis is performed by using library of designed and characterized storage elements as FIFO, LIFO and Registers (see Figure 1). The synthesis needs a specification of interleave scheme and a file containing user requirement (latency, throughput, communication interface, I/O parallelism ...) to generate a register transfer level (RTL) VHDL architecture.

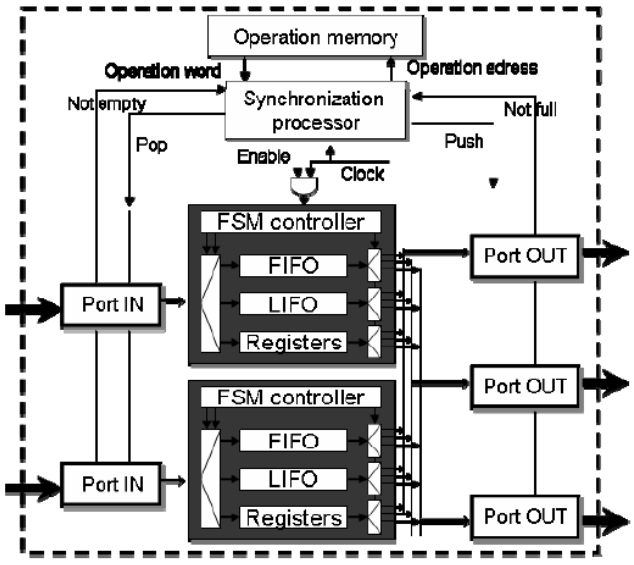


Figure 1. *STAR architecture [13]*

This is the design and integration method under I/O and timing constraints as described in[16]. This solution allows the adaptation of the communications between computing elements; however it does not handle the scheduling problem, as the data sequencing is considered here as a constraint[12]. The approach described in [27] allows to slow down the clock frequency in some of the parts of the design, to decrease the complexity

of the clock-network, to reduce the number of long wires and to perform clock-gating. This is to reduce the power consumption of FPGA architectures.

2.2. Multidimensional Synchronous Dataflow Model

The Synchronous DataFlow (SDF) model introduced by Lee and Messerschmitt[23, 30] and its extensions [26, 33] are very popular for the design of digital signal processing systems, especially because of their formal properties, which allow deadlock detection, static schedulability and the possibility to model multi-frequency systems. Initially, the SDF model was proposed for single-dimensional signal processing but work has been done to extend it to multidimensional signals, which are found for example in image processing system[23]. Besides having a greater expressive power than SDF, multidimensional synchronous dataflow (MDSFD)[30, 15] can detect data parallelism in a system. This model gives the ability to implement nested resetable loops and facilites data parallelism exploitation.
As described in [30], scheduling tasks in MDSDF consists in repeated calculation and schedule generation; it uses the same algorithms as in the SDF model. These algorithms are presented in [4].

2.3. Latency Insensitive Design

The theory of Latency Insensitive System (LIS) was introduced by Carloni [8] for the design of complex systems by assembling flexible components. It is based on the assumption that communications between the components is done by means of channels having *zero delay*. This theory has enabled the development of a new design methodology for large systems on a chip. The main idea for communication is pipelining : critical interconnections are partitioned into interconnections whose durations meet the time constraints imposed by the clock period by inserting logic blocks called relay stations. The relay stations are responsible for traffic regulation and deadlock management.

Carloni et al [8] propose an approach where a wrapper provides all signal activations that are necessary for a component. The wrapper here is composed of a combinational logic. The component is activated only if all its input are valid and if one has enough memory to store the results of its next execution.

As shown in Figure 2, receiving the signal `"valid"` for each input port means that the data is available and the signal `"ready"` for each output port means that the output device is ready to receive the results. If all these signals do not take the value "1" simultaneously, then a signal `"stall"` is sent to all input/output and the input/output process is interrupted thus freezing the component through a clock enable signal.

Bomel et al [5] propose to use a specific processor that reads and executes cyclically the operations stored in a memory. This Synchronization Processor (SP) communicates with the LIS ports through FIFO like signals. These signals are formally equivalent to `valid`, `ready` and `stall` of [8] and [39]. The SP model is specified by a Finite States Machine (FSM) with three states : a reset state at power up, an operation-read state, and a free-run state. This FSM run concurrently with the component and contains a data path. Operation's format is the concatenation of an input-mask, an output-mask and a free-run cycles number. The masks specify respectively the input and output ports the FSM is sensible to. The number of running cycles represents the number of clock cycles the component can execute until the next synchronization point. To avoid unnecessary signals and

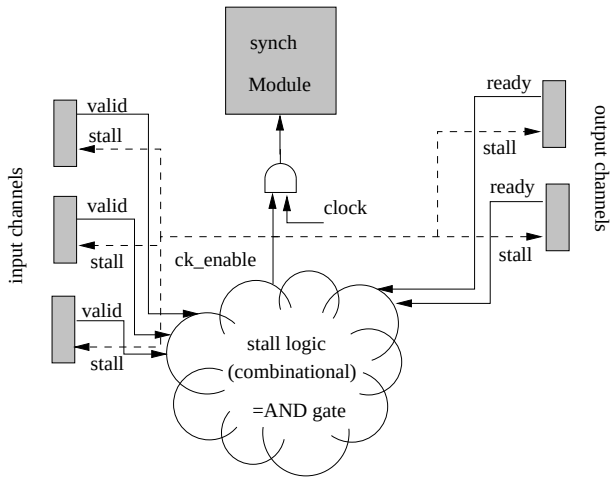


Figure 2. *Wrapper internal logic with combinational logic [38]*

save area, the memory is an asynchronous ROM (or SRAM with FPGAs) and its interface with the SP is reduced to two buses : the operation address and operation word (as shown in Figure 3). The execution of the program is driven by an operation *"readcounter"* incremented modulo the memory size.

Casu and Macchiarulo [9] show that, in order to reduce the hardware cost due to the use of component activation signals, it is possible to replace the synchronization protocol used in the previous approaches by a periodic scheduling algorithm applied to the component's clock. The component activation control is implemented with Shift registers. This approach relies on the hypothesis that there is no irregularity in data streams : it is never necessary to randomly freeze the component.

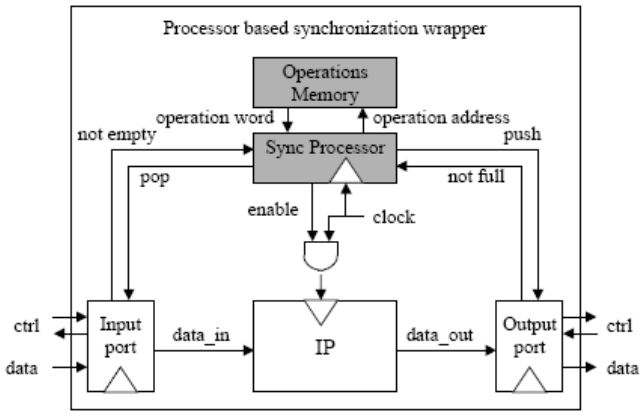


Figure 3. *Wrapper with Synchronization Processor [5]*

2.4. Multi-clock Architecture

Singh et al[39, 38] proposed an extension of Carloni's approach where it is assumed that components are frozen if no data is available for the next execution. So, instead of generating one stall signal for all input and output, it generates one signal for each port. In this approach the combinatorial logic that drives component clocks is replaced by Mealy type FSM as shown in Figure 4. This FSM tests the state of only the relevant input and output at each cycle and drives the component clock only when they are all ready. The approach can be implemented only if one disposes of input/output scheduling which proves that components communication behavior is cyclic and not data-dependant. Singh et al[38] extended point-to-point communication to more general communication topologies (fork, joint) and proposed an adaptation of the basic LID approach to multi-clock Latency-Insensitive Architecture.

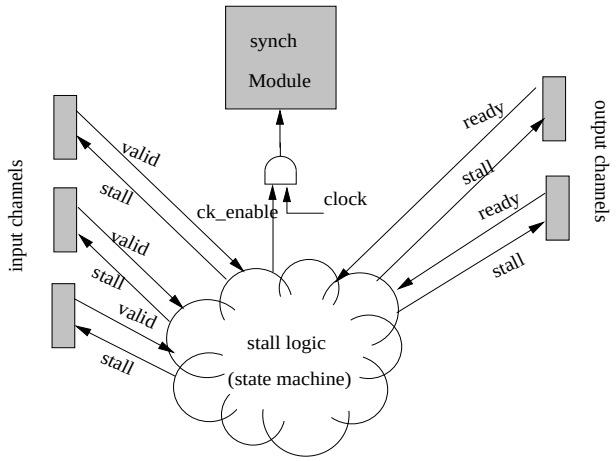
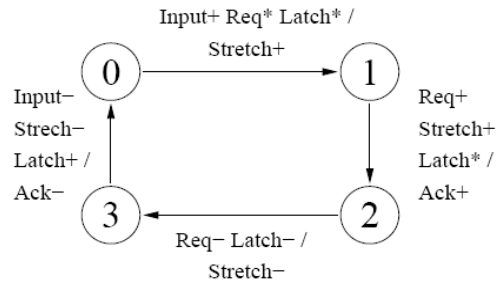


Figure 4. *Wrapper internal logic with State machine [38]*

Bormann et al [7] proposed a method for creating Globally Asynchronous Locally Synchronous (GALS) circuits. Each local synchronous module is surrounded by an Asynchronous Wrapper which provides an asynchronous interface to an otherwise module. Each data exchange is accompanied by a request-acknowledge pair of handshake signals so that it is unnecessary to consider the communication delay between blocks. The authors use a four-phase bundled data handshake protocol. There are four events per handshake cycle : `Req+`, `Ack+`, `Req-` and `ack-` (see Figure 5) and data is guaranteed to be valid when a `Req-` occurs and may change at any time after `ack-`. The wrapper synchronizes the asynchronous date to the local clock by stretching or pausing the clock.

2.5. Stream Programming

Stream programming expresses the parallelism inherent in a program by decoupling computation and memory accesses. The explicit parallelism and locality of data in a stream program makes it easier to compile efficiently using standard compiler optimiza-



$$Ack = Req*Stretch + Input*Ack + \overline{Latch}*Ack$$
$$Stretch = Req*Input + Input*\overline{Ack} + Latch*Stretch$$
$$Latch = \overline{Req} * Ack$$

Figure 5. *Bormann/Cheung wrapper scheme and boolean equations [7]*

tions. Initially targeted only for media/regular application, stream programming have evolved into a more general applications such as Fluid dynamic, sparse matrix computation or image processing. Programming streaming style can be summarized in three steps :gather, operate, and scatter. Using Direct Memory Access (DMA) units, the data is first gathered in bulk from arbitrary memory locations in main memory (MM) into local memories (LM) also called Stream Register Files (SRF). Then the data stream from the local memory is directly operated upon by one or more kernels, where each kernel is comprised of several operations. Finally, the output is scattered back in the main memory. In essence, a stream program decouples computation and memory accesses by boosting the memory reads before the computation, and postponing writes of the live data to memory after the computation. Stream programming, therefore, converts the memory latency problem into a memory bandwidth problem [19, 42]. The gathers and scatters of data may be from or to sequential, strided, or random locations in an array depending on the type of application.

Stream program is first compiled into an abstract machine model called Stream Virtual Machine (SVM) containing processors(control processor, computational/kernel processor, DMA units) and memory (global memory, SRF, local registers). The computational model can be described as follows[19]:

– The control processor schedules computational kernels and bulk memory operations on kernel processors and DMA units.

– The DMA units load the stream of data needed by computational kernels from the main memory to the SRF.

– The computational kernels work with the data in the SRF, using local registers to store temporary and intermediate results and writing live data in the SRF.

– After all the computational kernels having producer-consumer locality have finished computation, the DMA units write the live data back to the main memory.

Once generated, the SVM is compiled using a machine specific compiler, to the underlying stream architecture. While compilers do not have any control on processor caches, SRF can be sized to fit into processor caches, thus guarantying that the processor will bring into cache the SRF and does not write it back or replace it until the computational

kernel has finished its execution. Since computational kernels only access data store in the SRF, they don't suffer cache misses.

```
for j=1:n,
   for i=1:n,
     sum=0;
     for k=1:n,
       sum=sum+A(Arindex(i),Acindex(k))*B(Brindex(k),Bcindex(j));
     end;
     D(Drindex(i),Dcindex(j))=sum+C(Crindex(i),Ccindex(j));
   end;
end;
```

Figure 6. *MATLAB code for computing matrix-matrix product update D=A*B+C*

As an example, Consider the MATLAB code segment in Figure 6 for computing a matrix-matrix product update A*B+C; such a code will not parallelized by a standard Fortran and C compilers because of the index vectors used to access the elements of the arrays A, B, C and D. Stream programming will first transform the code into a form suitable for parallelism.

```
Kernel_dot(as, bs,  sum, n)
  sum=0;
  for i=1:n,
      sum=sum+as(i)*bs(i);
  end;

Kernel_matvect(bs, ys,  A(Arindex(1 .. n),Acindex(1 .. n)), Acindex, n)
     for i=1:n,
       /* gathers the Rrindex(i)-th row of A into a regular vector as*/
         Streamgather(as, A(Arindex(i),Acindex(1 .. n)), Acindex);
         Kernelcall("Kernel_dot",as, bs, sum, n);
         ys(i)=sum;
     end;

Kernel_vectadd(ys, cs, ds, n)
     for i=1 :n,
         ds(i)=ys(i)+cs(i) ;
     end ;
```

Figure 7. *Kernels codes:*

The k-th loop is an inner product ($a^T b$) of the Arindex(i)-th row (A(Arindex(i), Acindex(1..n)) of A and the Bcindex(j)-th column (B(Brindex(1..n),Bcindex(j)) of B; let assign this computation to kernel Kernel_dot. The i-th loop is a matrix-vector product update D(Drindex(1..n),Dcindex(j))=A*B(Brindex(1..n),Bcindex(j))+C(Crindex(1..n),Ccindex(j)). Let break it up into two kernels: the matrix-vector product ys= A*B(Brindex(1..n),Bcindex(j)), assigned to kernel Kernel_matvect and the vector addition ds = ys+C(Crindex(1..n),Ccindex(j)), assigned to kernel Kernel_vectadd. The codes for these kernels are given in (Figure 7).

The SVM code for the computation is as follows(Figure 8):

```
for j=1..n
   /* gathers the Bcindex(j)-th column of B into a regular vector bs*/
    Streamgather(bs, B(Brindex(1..n),Bcindex(j),Brindex);
   /* gathers the Ccindex(j)-th column of C into a regular vector cs*/
    Streamgather(cs, C(Crindex(1..n),Ccindex(j),Crindex);
    Kernelcall("Kernel_matvect",bs, ys,  A(Arindex(1 .. n),
                Acindex(1 .. n)), Acindex, n);
    Kernelcall("Kernel_vectadd",cs, ys, ds, n);
   /* scaters a regular vector ds into the Dcindex(j)-th column of D */
    Streamscater(D(Drindex(1:n),Dcindex(j)),ds, Drindex);
end
```

Figure 8. *SVM computation code*

Stream programs can be naturally represented as a graph of independent actors that communicate explicitly over data channels. In general the input and output rates of actors are known at compiler time. It can be simply represented using Synchronous DataFlow (SDF) graphs [19]. The input and output to the computation kernels, and the dependencies between the kernels are explicitly indicated. For the example above, the SDF graph is given in Figure 9. Gummaraju et al describe in [19] the procedure for executing a stream programme on a stream processor, the stream execution model and methodology to mapping the stream execution model on general purpose processor. The interest in streaming applications has spawned a number of programming languages that target the streaming domain, including StreamIt [42], Streamware [20], StreamC/KernelC [22].

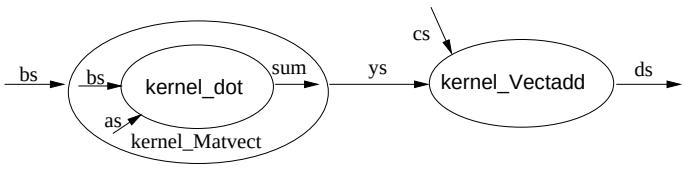


Figure 9. *SDF graph of a stream program of the matrix-matrix computation. The Kernel_matvect receives* `bs`*; stream loads* `as` *and produces* `ys`*. It uses Kernel_dot, which receives* `bs` *and* `as` *and produces sum. The Kernel_vectadd receives* `cs` *and* `ys`*, and produces* `ds`*.*

3. New Integration Approach

Our approach to integrating components consists of 3 main steps, namely:

– The system specification: behavioural description of the system as SDF graph and checking the system hardware feasibility.

– The automatic synthesis: this stage takes as input an ALPHA description of the system and automatically computes the scheduling, determines logical clocks period, and generates the VHDL controller code and the VHDL code of the whole system.

– Components Assembling: the last phase combines the codes obtained from the previous step with the VHDL code of the various components. The final code can thus be submitted to a physical synthesis tool such as Xilinx ISE or Quartus of Atera.

3.1. System Specification

The system is modeled using the synchronous dataflow graph. This model is more suitable for periodic multi-frequency applications. It offers to the designer the possibility to verify the feasibility of the specification before starting the synthesis.

Consider for illustration the system described by the SDF graph in Figure 10. This exemple consists of elementary hardware components (rectangles) `cal1`, `cal2`, `cal3` and synchonization components (circles). The nodes represent the treatments and the arcs are communication (FIFOs) which are bounded and unidirectional memory; a value on the arcs are the number of data produced or consumed at each iteration.

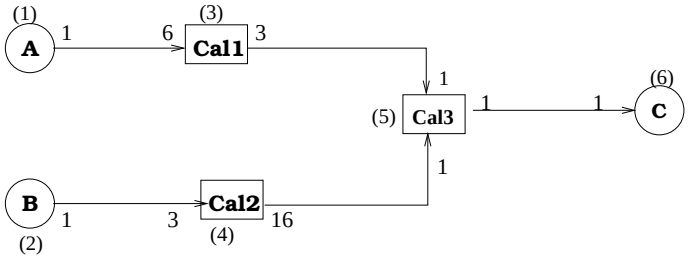


Figure 10. *SDF graph Sample*

This system can be described by a system of linear equations also referred to a balance equations. A balance equation is based on the following condition: when iterating, the amount of data produced by the source block is equal to the amount of data consumed by the target block. for the above exemple the description is as follows :

$$\begin{aligned} x_1 - 6x_3 &= 0 \\ x_2 - 3x_4 &= 0 \\ 3x_3 - x_5 &= 0 \\ 16x_4 - x_5 &= 0 \\ x_5 - x_6 &= 0 \end{aligned}$$

where the $x_i's$ are the frequencies at which the nodes must be invoked.

Finding a solution to this systems means ensuring that the SDF graph can be executed with a limited amount of memory.

We can write the system in the matrix form $\Gamma x = 0$ or

$$\begin{pmatrix} 1 & 0 & -6 & 0 & 0 & 0 \\ 0 & 1 & 0 & -3 & 0 & 0 \\ 0 & 0 & 3 & 0 & -1 & 0 \\ 0 & 0 & 0 & 16 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Γ is called topological matrix : the columns represent nodes and the lines represent arcs. $\Gamma(i, j)$ represents the amount of data produced or consumed by the node j on arc i each time it is invoked. This quantity is positive or negative if information is produced or consumed. It is zero if node j is not connected to arc i.

$\Gamma \in \mathbb{R}^{s-1 \times s}$ and $rank(\Gamma) = s - 1$; since $rank(\Gamma) + null(\Gamma) = s$ is follow that $null(\Gamma) = 1$ and the null space is reduced to a single element which is the vector solution of the system $\Gamma x = 0$. This is the necessary condition for the existence of a statical and periodical system scheduling. The components of this vector are frequencies indicating how often a giving node of the graph should be invoked during a period T for the system to remain consistent [23, 10]. For the above example the unique solution is,

$$x = \begin{pmatrix} 96 \\ 9 \\ 16 \\ 3 \\ 48 \\ 48 \end{pmatrix}.$$

The period being defined as the inverse of the frequency, one can deduce the period of each node by taking the inverse of its frequency. To reduce these periods to integer values, they are multiplied by the lowest common multiple (LCM) frequencies. The resulting vector contains the logic clocks period denoted by *CE*. It is worth pointing out that if specification leads to a solution with very high clock frequency, it may not possible to generate an architecture for the system.

After the system specification, one can verify the assembly conditions by checking the existence of a periodic schedule using this topological matrix. Lee et al propose in [23] some verification algorithms. If the assembly is possible, we move to the next step; if not we must review the specification.

3.2. Automatic Synthesis using MMALPHA Environment

In this section we present an ALPHA description of parallel data stream application whose modules run at different frequencies. Those applications called Multirate Parallel Alpha DataFlow Systems (MP&DFS) were introduced by [11, 10].

Notice that, an ALPHA structured system is an ALPHA system that uses subsystems or components.

3.2.1. Hardware Component Modeling

The components are *synchronous elements*, and their synchronization is based on a fundamental clock signal denoted by `clk`. Each component is physically synchronized with a clock validation signal *clock-enable* denoted by *CE* [35]. For the system in Figure 10 we have

$$\hat{x} = \begin{pmatrix} 9 \\ 96 \\ 54 \\ 288 \\ 18 \\ 18 \end{pmatrix}$$

In other words, all registers of the component are loaded under the control of this *CE* signal.

So, each elementary hardware component is considered as *combinational element component*, computing stream in term of input streams and storing the result in a register. As illustrated in Figure 11, this component is synchronized by a clock-enable signal, to operate at a period of 9 clock cycles, and with an initial shift of 2 clock cycles or with period 18 without shift.

We can see that, by changing the pattern of the *CE* signal, we may use the same component with different periods and different shifts: indeed, all registers of the component operate at a virtual clock whose period and shift are specified by *CE*. The combination of `clk` and *CE* is denoted by a pair (λ, φ) where λ represents the period of the signal and φ its initial shift.

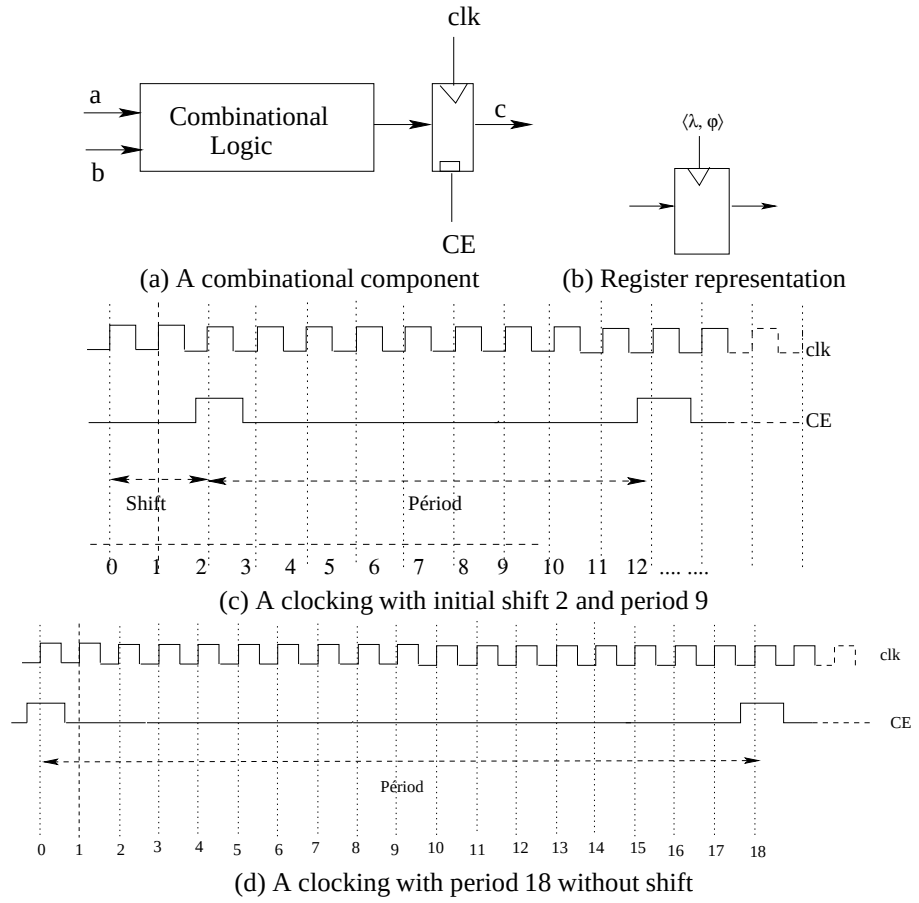


Figure 11. *A combinational component (a) and its synchronization can be (c) using a clock-enable signal with period 9 and initial shift is 2; or (d) with period 18 without shift. (b) shows the representation of a register with period λ and shift φ.*

3.2.2. System Specification using ALPHA

ALPHA programs are functional (declarative) description that operate on variables which are functions from a polyhedral domain to some type of data. This is best explained by the example of Figure 12 which presents a sketch of the system describe in Figure 10.

Line 1 is the declaration of a *system* of ALPHA equations. Lines 2 and 3 define the input of this system, namely, the `A` and `B` variables. Both variables correspond to a semi-infinite sequences of 2-bit signed integers, that we call *streams* for the purpose of this paper. Similarly, line 6 defines the output `C`. Lines 8 and 9 contain the declaration of local variables. Lines 11 and 13 are simple assignment of `A` and `B` to the local stream `Amirr` and `Bmirr` which are stored in the registers (lines 12 and 14) before being used by the modules `Cal1` and `Cal2` respectively. Line 12 makes use of an ALPHA subsystem, here

representing a register file of length 1. Lines 15, 16 and 17 instantiate components `Cal1`, `Cal2` and `Cal3` respectively.

Each `use` statement allows one to instantiate the definition of another, pre-defined, ALPHA subsystem.

```
1 system Compute
2   (A : {i|0<=i} of integer[S,2];
3    B : {j|0<=j} of integer[S,2])
4 returns
5   -- output data and control
6   (C: {i|0<=i} of integer[S,2]);
7 var
8   out_cal1, out_cal2, in_cal3: {i|0<=i} of integer[S,2];
9  Amir, AmirReg, Bmir, BmirReg, mul, Cmir: {i|0<=i} of integer[S,2];
10 let
11   Amir = A;
12   use registerFile[1] (Amir) returns (AmirReg);
13   Bmir = B;
14   use registerFile[1] (Bmir) returns (BmirReg);
15   use Cal1(AmirReg) returns (out_cal1);
16   use Cal2(BmirReg) returns (out_cal2);
17   use Cal3 (out_cal1,out_cal2) returns (in_cal3);
18   use registerFile[1] (in_cal3) returns (Cmir);
19   C = Cmir;
20 tel;
```

Figure 12. *The* ALPHA *Program of the example of figure 10*

3.3. Scheduling

In [11] it is explained how to schedule stream components, but we summarize here in order to illustrate its power. The method that we use has two steps: first, find out the *period* of each subsystem, then compute an overall *affine* schedule for the entire system. Consider a subsystem `S` with input `I` and output `O` (to simplify the matters, we assume only one input and one output). Assume that this subsystem admits an integral schedule of the form $T_{\mathtt{I}}(i,...) = i + \alpha_{\mathtt{I}}$, and $T_{\mathtt{O}}(i,...) = i + \alpha_{\mathtt{O}}$, where i denotes the index of the input and output streams. (Notice that streams of values are multi-dimensional, and the other dimensions of the streams are represented by dots.) Such schedules are called *data-flow* schedules, since they are monotonically increasing functions of the first index `i` of the stream.

In other words, we assume that whenever an output $\mathtt{O}(i,...)$ depends on an input $\mathtt{I}(j,...)$ through the equations which define `S`, then $T_{\mathtt{O}}(i,...) > T_{\mathtt{I}}(j,...)$, and moreover, we assume that T is non-negative.

Note that for all positive integers λ, $\lambda T_{\mathtt{I}}$ and $\lambda T_{\mathtt{O}}$ are also a valid schedule for this system. Indeed, if $T_{\mathtt{O}}(i,...) > T_{\mathtt{I}}(j,...)$, then $\lambda(T_{\mathtt{O}}(i,...)) > \lambda(T_{\mathtt{I}}(j,...))$, since T is nonnegative for all i. We call this a λ-*slow* version of S.

To implement such a system, it suffices to clock the corresponding hardware subsystem with a λ times faster clock, that enables to have valid signal every 2 clock tick. Note that, with this implementation, *the number of registers is unchanged* inside the hardware subsystem.

In general, a *dataflow* ALPHA *schedule* is a linear function T which assigns to each variable A of an ALPHA data-flow system a schedule of the form $T_{\mathtt{A}}(i,...) = \lambda_{\mathtt{A}} i + \beta_{\mathtt{A}}$, with the constraints that:

1) if A depends on B through a regular data-flow system, then $\lambda_{\mathtt{A}} = \lambda_{\mathtt{B}}$;

2) if A depends on B through a K-up-sampler, then $\lambda_{\mathtt{B}} = K\lambda_{\mathtt{A}}$;

3) if A depends on B through a K-down-sampler, then $\lambda_{\mathtt{B}} = \frac{\lambda_{\mathtt{A}}}{K}$;

4) $\beta_{\mathtt{A}} = \lambda_{\mathtt{A}}\alpha_{\mathtt{A}}$, for all A.

The λ's are positive integers, and the α's are non-negative integers. This definition ensures that all elements of the system are scheduled with buffers whose length don't depend on i. The conditions on the λ coefficients give a system of homogeneous integral equations of a special type, since all equations have the form $\lambda_1 = K\lambda_2$, where K is a strictly positive rational number.

In [23], it is explained how such a system can be solved in time linear in the number of equations. Once the periods are found, they can be combined with the schedule constraints that correspond to each component (i.e. ALPHA subsystem), and any solution provides a scheduling for the whole system. A structured polyhedral scheduling approach can be taken to solve this problem, as explained in [11].

4. Wrapper Architecture and Components Assembling

4.1. Wrapper Architecture

The method we propose is to add to the hardware description of each component a *wrapper* that allows this component to be executed according to its imposed schedule. Figure 13 illustrates the model of the wrapper we have developed. Besides its data inputs and outputs, a component has a clock signal (`clk`), a general clock-enable signal (*CE*) and a reset signal (`rst`).

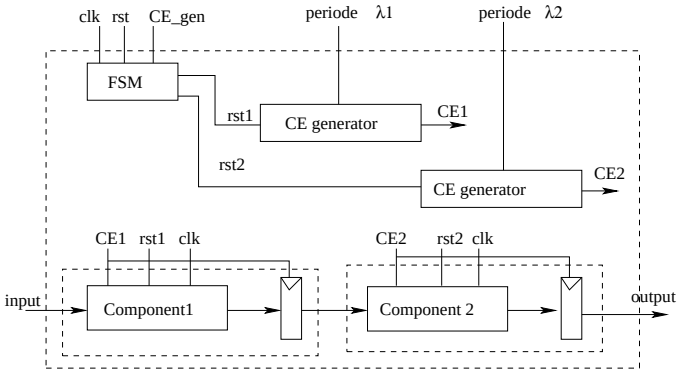


Figure 13. *Wrapper architecture. In this representation we have two components which use clock-enable signals `CE1` and `CE2` generated with periods λ_1 and λ_2 respectively.*

The wrapper includes also clock-enable generators, one for each period used in the system. These clock-enable signals are used to generate the various delayed clock-enables, by means of delay lines. Clock-enable generators are simply modulo counters, the register of which is triggered by the general clock-enable signal of the wrapper (see Figure 14).

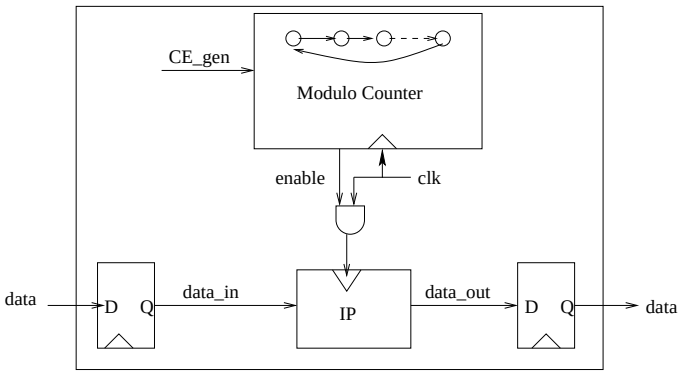


Figure 14. *Wrapper process model*

In order to enhance the synchronization of the whole system with its environment, a further activation signal has been added, this is a general clock enable signal. We use for this purpose a model based on FIFO queue. This type of interface is widely used in the systems that operate on data streams input and produce a data stream as output. According to the status of FIFO, the general clock enable signal can freeze or not the system. This signal denoted by `CE_gen` is generated according to equation:

```
enable = (NOT(full_fifo_out) AND NOT(empty_fifo_in) )
```

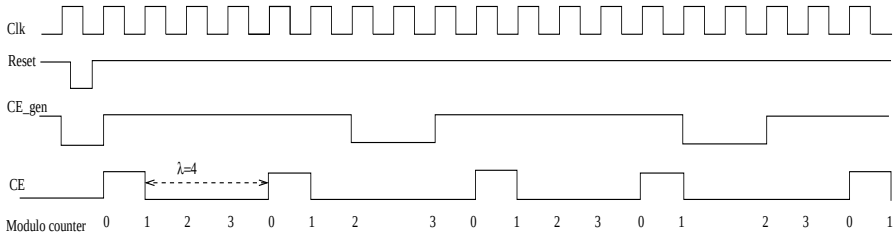


Figure 15. *Enable signal* $CE_{\lambda i}$ *whith* $\lambda = 4$.

```
1 ENTITY FSM IS
2  PORT(
3    clk : IN STD_LOGIC;
4    CE_gen : IN STD_LOGIC;
5    rst : IN STD_LOGIC;
6    reset0 : OUT STD_LOGIC;
7    reset1 : OUT STD_LOGIC);
8   END FSM;
9 BEGIN
10 ---- Synchronous reset process
11 PROCESS(rst,clk)
12 BEGIN
13   IF clk = '1' AND clk'event THEN
14    IF CE_gen = '1' THEN
15      IF rst='0' THEN
16        curstate <= state0;
17        count <= 0;
      ..............................
      ..............................
..............................

18 END archiOfFSM;
```

Figure 16. *Entity FMS generated by* MMALPHA

It is part of activation signals of each component. Thus the enable input signal of the various components is a combination of $\texttt{CE}_{\lambda i}$ and `CE_gen` according to Figure 15 with $\lambda = 4$.

4.2. Components Assembling

After the system scheduling, the MMALPHA environment produces two VHDL codes, the controller code and the all system code. For illustration consider the application de-

scribed in Figure 12 we obtain : (i) **the controller code** which has two entities; the entity FMS (Figure 16) that takes as inputs clk, rst and CE_gen generated as described in section 4. It produces two outputs because we have only two logical clocks in this case: the output reset0 is used by the component FFT as shown at lines 21 and 22 of Figure 18, and the output reset0 used by the component PeriodEnable128 (see Figure 17). The role of PeriodEnable128 is to produce the periodic validation signal CE with period 128.

(ii) **The VHDL code of the all system** is presented in Figure 18, this code has several parts: Lines 4 to 12 correspond to the system declaration(the entity here is TwoFFT); lines 14 to 15 are the declaration of the flexible component (IP), we have here one component, the FFT component. The components Registersenable1_1 is register (lines 16 and 17). Line 21 to 24 are the instantiations of all the components with the appropriates signal.

```
1 ENTITY PeriodEnable128 IS
2  PORT(
3    clk : IN STD_LOGIC;          -- global clock
4    ceGen : IN STD_LOGIC;       -- general clock enable signal
5    rst : IN STD_LOGIC;          -- reset signal
6    periodicGe : OUT STD_LOGIC     -- periodic clock enable );
7  END PeriodEnable128;
.........................
.........................
8 PROCESS( clk )
9  BEGIN
10    IF rising_edge(clk) THEN
11      IF ceGen='1' THEN
12        IF rst='0' THEN        -- enable is 0 when rst
13          counter <= "0000000";
       ..........................
14  periodicGe <= '1' WHEN (counter = "0000000" AND ceGen = '1') ELSE '0';

15 END archiOfPeriodEnable128;
```

Figure 17. *Entity Periodic enable generated by* MMALPHA

```
1 -- VHDL Model Created for "system TwoFFTs"
2 -- 28/12/2010 16:19:56.283228
3 -- Alpha2Vhdl Version 0.9
.....................................
4 ENTITY TwoFFTs IS
5 PORT(
6   clk: IN STD_LOGIC;
7   CE : IN STD_LOGIC;
8   Rst : IN STD_LOGIC;
9   A : IN SIGNED (1 DOWNTO 0);
10  B : IN SIGNED (1 DOWNTO 0);
11  C : OUT SIGNED (1 DOWNTO 0));
12 END TwoFFTs;
     ..............
     ..............
13 -- Insert missing components here!
14 COMPONENT FFT IS
15  PORT(
          ..............)
16 COMPONENT Registersenable1_1 IS
17  PORT(
     ..............)
18 -- Controller
19 COMPONENT PeriodEnable128 IS
20  PORT(
..............)

21   G1 : FFT PORT MAP (clk, enable1_1, reset1, AmirReg, fft1);
22   G2 : FFT PORT MAP (clk, enable1_1, reset1, BmirReg, fft2);
   ............................
23   G11 : Fsm PORT MAP (clk, CE, rst, reset0, reset1);

24   G14 : PeriodEnable128 PORT MAP (clk, CE, reset0, enable128);
.....................................
25 END behavioural;
```

Figure 18. *VHDL Code of the all system generated by* MMALPHA

4.3. Time Model

The interface between our system model and the physical world constitutes a simple model of time, whose characteristics can be summarized as follows:

– Our systems are *synchronized*, in the sense that computations are triggered by a common background clock.

– The time behavior of our components is *affine-periodic*. Each component in the system is supposed to execute at periodic instants of time, after an initial shift. This model does not support exceptional events, as true reactive systems do, unless these events were watched at regular, periodic instants of time.

– Our model of time supports some forms of parallelism. Indeed, each component can be an affine-parameterized assembly of parallel components, either pipeline (or systolic) components, or true parallel systems operating on vectors of data.

– The systems that we consider are *time-delayable*. This means that a component can be scheduled several clock ticks later or earlier, without affecting the semantics of the whole system. In other words, provided the final implementation of the system meets the dependence constraints expressed by the initial description, we consider that its behavior is correct. This contrasts with the underlying hypothesis of *synchronous languages* such as Signal [40], where the description of the flows of data represents a strict specification of the behavior of the physical constraints put on the system, and therefore, cannot be relaxed.

– Components are *slowable*. This means that they can operate at any sub-clock of their fundamental clock, under the control of the clock-enable signal. This property has some relationship with *delay-insensitivity* where components can be executed even in the presence of some delays in their inputs.

– Time is *stretchable*. As summarized in figure 19, the clocks used here are logical clocks. They are based on a global clock and activation signals which depend on the period (stretch) of each component.

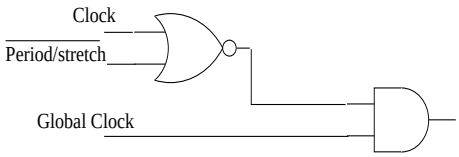


Figure 19. *A stretchable clock made using external clocks.*

5. WCDMA Implementation

These methods were implemented as part of the MMALPHA environment. Components are defined by means of ALPHA systems and can be synthesized using the MMALPHA software.

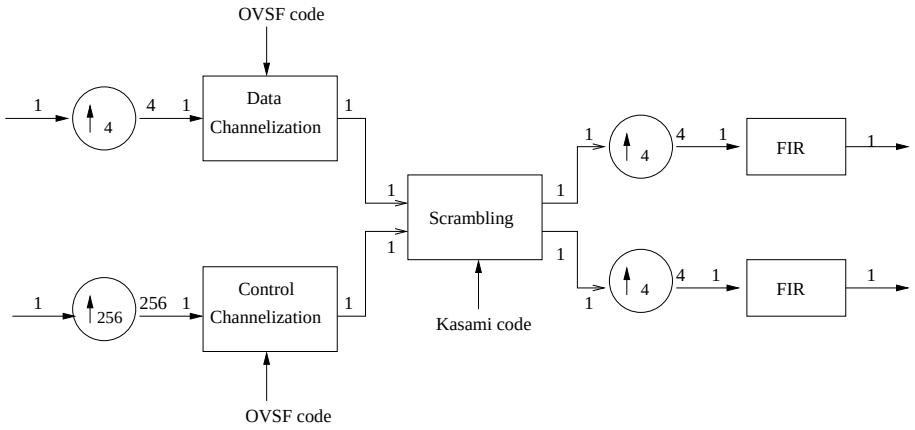


Figure 20. *Graph of a simplified* WCDMA *emitter;*

To evaluate the methodology that we have presented, we have realized the WCDMA emitter. Figure 20 represents a simplified version of a WCDMA emitter[14]. It consists of elementary hardware components (rectangles), synchronization components (circles), connected together with arcs. Following the SDF actor notation[4], an arc carries at its origin the number of values produced by a component during one of its execution, and at its extremity, the number of values consumed by a component during one execution of the receiving component.

The inputs to this WCDMA graph are a data signal and a control signal. Frequencies of data and control are different: there is one control value for 64 data values. The WCDMA emitter contains, from the left to the right:

– Two up-sampling components, one for the data signal, and one for the control signal (round up-samplers). Their role is to reduce the signal frequencies to the WCDMA standard frequency which is 3.84MHz, by spreading by a factor of 4 for data and 256 for control.

– Channelization components, whose role is to multiply the signals previously up-sampled by orthogonal codes of variable length OVSF (Orthogonal Variable Spreading Factor). Spreading transforms every bit into a given number of chips, hence increasing the bandwidth. For the sake of simplicity, we do not represent here the memory system for storing and supplying these data and control OVSF codes.

– A scrambler, which is a complex multiplication of signals by Kasami codes; this operation allows the system to distinguish the information coming from different terminals, mobile and base stations.

– Up-sampling by a factor of 4 before filtering.

– Two Finite Impulse Response filters (FIR) which aim to cancel inter-symbol interference between different information.

The outputs of this system consist of the two signals emitted by the filters, at the same frequency.

These components are executed at different frequencies. Taking as reference the filters which are executed at the highest frequency, the clock period of the filters is 1, for the scrambling and channelization the period is 4, the period of the input data is 16 and that

of the input control is 1024. The periods are calculated as described in section 3 and represented in the Figure 21. In this Figure the components are represented in terms of combinational element (circle) and register as described in section 3.2.

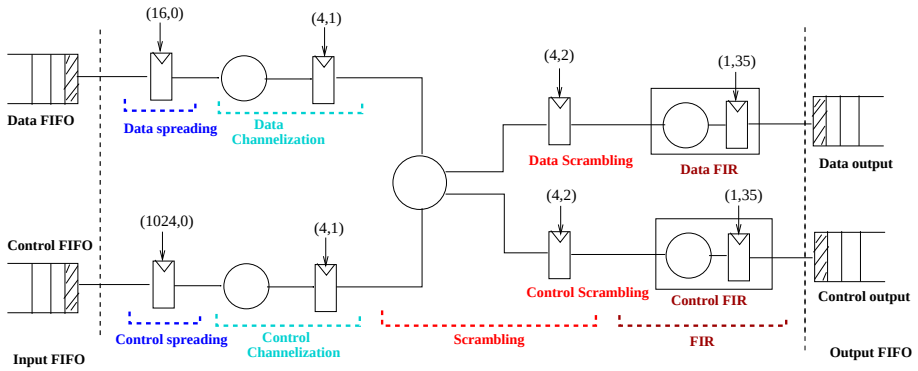


Figure 21. WCDMA *emitter representation after scheduling process*

We also know the detailed schedule of the components, in particular the frequency of their inputs and outputs and when needed a reference starting time for their controller. In this particular example, only the filters need a controller, which explain the $(1, 35)$ for `Data FIR` and `Control FIR` in Figure 21. So, the filters behavior must therefore be taken into account in the overall controller that we generate. The wrapper, and the controller were generated automatically.

The generation of the complete VHDL description by MMALPHA takes $1.10s$ on a current MacBook laptop, including the scheduling that is done in $0.03s$ using MATHEMATICA.

The whole system was simulated with the Mentor Graphics tool Modelsim SE-2d [43] for the functional verification. The hardware synthesis of each component as well as the entire system were made within the Xilinx ISE environment [28]. Table 1 presents the results of the synthesis for the WCDMA emitter produced by this method. For each component, the number of slices, multiplier-accululators (MAC), latches and the estimated clock frequency are given. This example shows that the area of the wrapper is negligible, compared to the components of the emitter. Moreover, it has no influence on the estimated frequency of the whole architecture.

Component	#Slices	#MAC	#Latch	Frequency (MHz)
FSM	24	0	34	-
Logical clock 4	2	0	2	-
Logical clock 16	10	0	13	-
Logical clock 1024	10	0	13	-
Wrapper (total)	41	0	53	415,6 MHz
Emitter (total)	1962	68	7528	129,5 MHz

Table 1. *Results of the synthesis for the upstream* WCDMA *emitter on a Xilinx Virtex4* FPGA

6. Comparison with Related Work

In this section we compare our work with some related work.

– **Synthesis under constraints:** In our approach we implicitly take into account the size of storage elements because, in the behavioural description of the system we know how many data is produced or consumed by each block. So it is possible to guarantee as describe in section3.1 that system can be executed with a limited amount of memory. This solves the problem of memory size required to store data. Unlike the methods of synthesis under constraints ([12, 16]) we consider that data are produced and consumed in the same order.

– **Multidimensional Synchronous Dataflow Model:** MDSDF shares a lot of properties with the model presented in this paper. The calculation of the periods of our components appears as a transposition, in the domain of recurrence equations, of the balance equations of SDF. But in our approach the expression of ALPHA programs is much more suited to the description of parallel systems, and closer to a loop-like description of calculations.

– **Latency Insensitive Design:** These models generate a significant additional hardware resource in order to take into account the different control signals. Our approach, just like the extension proposed by Casu and Macchiarulo, does not use activation signals (`"Stall"` or `read`) but periodic scheduling process. In addition, our model is suitable for modelling multi-frequencies systems where components are executed at different frequencies.

– **Multi-clock Architecture:** similar to the approaches described in section 2.4, we propose a multi-clock system design approach with the advantage that scheduling allows us to know exactly at what time each component must be enabled. We do not need activation signals as in [7] and [38], thus reducing hardware cost due to the use of component activation signals. But we use the point-to-point communication and system throughput is limited by the throughput of the slowest synchronous module. We generate a systolic architecture which is a network of processors that rhythmically compute and pass data through the system. So the system globally synchronous.

– **Stream programming:** The stream programming is a programming paradigm that generate a Stream Virtual Machine (SVM) and ALPHA is a hardware synthesis language that generate systolic architecture. But despite this, they share a lot in common. They are based on the same model, the synchronous dataflow graph. Stream programming have been developed to handle systems computing on streams of data. The extension we added to ALPHA makes it able to generate architecture for stream applications. ALPHA can generate parallel SIMD architectures. In our approach the flexible components communicate through bounded FIFO and exploit the producer-consumer principle. The data produced are consumed directly without going through the main memory.

The originality of this work is the introduction of relationship between synchronous dataflow model the polyhedral model, thus enabling the use of very powerful techniques to detect, handle parallelism in stream applications and to manage the transformation of parallel programs to architectures. The systems that we obtain are *time-delayable* something that is not possible for those described using *synchronous languages* such as Signal [17, 3], which use *affine-periodic* time behavior also.

In our approach, we replace the clock based on the combinational logic in the initial approaches to LIS by finite state machine of Moore type that is much easier to implement than those used by [38, 39] which are Mealy machines. The complexity of the wrapper that we offer does not depend on the number of component computation cycles as in other approaches, but only on the number of ports; thus reducing the hardware cost.

7. Conclusion

We have presented a model for the automatic hardware synthesis of stream systems. A simple model of time was introduced, and its properties were shown to allow a detailed scheduling to be found automatically. A hardware implementation was presented, based on the addition of a simple wrapper.

Our work combines the synchronous data-flow approach and the polyhedral model to achieve a fully automated synthesis for some kinds of streams systems which are often found in Cyber physical systems. Our main contribution is a model that combines both stream processing and parallel management. Indeed, the polyhedral model is a very powerful framework for exploiting loop parallelism, either for programming parallel architectures or for the high-level synthesis of hardware devices.

Further research will aim at investigating the limits of our simplified model of time, both in terms of potential for tractable automatic methods, and in terms of interfacing with complex Cyber physical systems, where the interaction with the physical world cannot be represented only as *regular* streams of data.

8. References

[1] F. ABBES, E. CASSEAU, M. ABID, P. COUSSY, J.B. LEGOFF, "IP Integration methodology for SoC design ", *IN ICM04 International Conference on Microelectronics*, Tunis, dec, 2004.

[2] I. AUGÉ AND F. PÉTROT AND F. DONNET AND P. GOMEZ " Platform-Based Design From Parallel C Specifications ", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(12) pages 1811-1826*, 2005.

[3] L. BESNARD, T. GAUTIER AND P. LE GUERNIC, AND J. TALPIN "Compilation of polychronous dataflow equations ", *Synthesis of Embedded Software Springer*, 2010

[4] S. S. BHATTACHARYYA "Compiling Dataflow Program for digital signal processing ", *PhD Thesis, University of Califormia at Berkeley*, jully,1997.

[5] P. BOMEL AND E. MARTIN AND E. BOUTILLON, "Synchronous Processor synthesis for Latency Insensitive Systems ", *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, 2005.

[6] P. BOMEL "GAUT, High Level Synthesis for Digital Signal Processors: User's Manual, version 4.1 " *Laboratoire d'Electronique des Systèmes Temps RéelLESTER, University of southern Britany*.

[7] D. BORMAN AND P. Y. K. CHEUNG "Asynchronous wrapper for heterogeneous systems" *In proceedings of ICCD*, 1997.

[8] L. P. CARLONI AND K. L. MCMILLAN AND A. L. SANGIOVANNI-VINCENTELLI, "Theory of Latency-Insensitive Design ", *IEEE Transactions on Computer-Aided Design of Integrated Circuits ans Systems*, num. vol. 20, NO. 9, sptember, 2001.

[9] M. R. CASU AND L. MACCHIARULO "Adaptive Latency-Insensitive, Globally Asynchronous, Locally Synchronous Design and Test ", *IEEE Design & Test of Computers*, 2007.

[10] A. CHANA AND P. QUINTON "Intellectual Property (IP) Integration Approach for Data-Flow parallel embedded Systems ", *Proceedings of Fourth International IEEE EAI Conference on e-Infrastructure and e-Services for Developing Countries* num. November, 2012.

[11] F. CHAROT AND M. NYAMSI AND P. QUINTON AND C. WAGNERA "Modeling and scheduling Parallel DataFlow systems using structured systems of Recurrence Equations ", *Procedings of the 15th IEEE International Conference on Aplication-Specific system, Architectures and Processors(ASAP'04)*, 2004.

[12] C. CHAVET "Synthèse automatique d'interfaces de communication matérielles pour la conception d'applications du domaine du traitement du signal", *Thèse, Université de Bretagne Sud*,2007.

[13] C. CHAVET AND P. COUSSY AND P. URAD AND E. MARTIN "A Design Methodology for Space-Time Adapter ", *IEEE/ACM Great Lakes Symposium on VLSI, March*,2007.

[14] H. -H. CHEN "The Next Generation of CDMA Technologies ", *John Wiley and sons*, 2007.

[15] M. J. CHEN AND E. A. LEE "Design and Implementation of a Multidimensional Synchronous Dataflow environment ", *IEEE Asilomar Conference on Signal, System and computer*, 97.

[16] P. COUSSY AND E. CASSEAU AND P. BOMEL AND A. BAGANNE AND E. MARTIN "A Formal Method for Hardware IP Design and Integration under I/O and timing constraints ", *ACM Transactions on Embedded Computing systems, vol. 5, N°1, page 29-53*,2006.

[17] A. GAMATIÉ AND T. GAUTIER "The Signal Synchronous Multi-clock Approach to the Design of Distributed Embedded Systems ", *IEEE Transactions on Parallel and Distributed Systems*, 2010

[18] A. GERSTLAUER ET AL "Electronic System-Level Synthesis Methodologies ", *IEEE Transactions on Computer-aides Design of Integrated Circuits and Systems*,2009.

[19] J. GUMMARAJU AND M. ROSENBLUM " Stream Programming on General-Purpose Processors ", *in Proceedings of the 38th annual international symposium on microarchitecture (MICRO-38), November 2005, Barcelona Spain*

[20] J. GUMMARAJUZ AND J. COBURNZ AND Y. TURNERX AND M. ROSENBLUM " Streamware: Programming General-Purpose Multicore Processors Using Streams " *ASPLOS'08, March 1, 2008, Seattle, Washington, USA.*

[21] S. GUPTA AND R. GUPTA AND N. DUTT AND A. NICOLAU " SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits ", *Kluwer Academic* 2004.

[22] B. KHAILANY ET AL " Imagine : Media processing with streams " *IEEE micro, 21(2) :35-46* 2001.

[23] E. A. LEE, D. G. MESSERSCHMITT "Static Scheduling of Synchronous DataFlow Programs for Digital Signal Processing ", *IEEE Transactions of Computers*, num. vol. C-36, page 135-140, 1987.

[24] E. A. LEE "Cyber physical systems : Design challenges ", *In International Symposium on Object/Component/service-oriented real time Distrubuted Computing(ISORC)*, May 2008.

[25] E. A. LEE AND S. A. SESHIA "Introduction to Embedded Systems - A Cyber-Physical Systems Approach ", *LeeSeshia.org*, 2011.

[26] EDWARD A. LEE AND THOMAS M. PARKS " Dataflow Process Networks ", *published in proceeding of the IEEE*, May, 1995.

[27] G. LHAIRECH-LEBRETON AND P. COUSSY AND E. MARTIN "Hierarchical and Multiple-clock Domain High-Level Synthesis for Low-Power Design on FPGA ", *Proceedings of FPL*, 2010.

[28] MENTOR GRAPHICS CORPORATE "http://www.model.com, Technical report ", *Modelsim, simulation environment*, 2012

[29] C. MAURAS "IRISA, Université de Rennes I ", *Définition de ALPHA: un langage pour la programmation systolique*,1989

[30] P. K. MURTHY AND E. A. LEE "Multidimensional synchronous dataflow ", *IEEE Transactions on signal processing*, num. vol.50, page2064-2079, august, 2002.

[31] H. NIKOLOV ET AL "Automated Integration of Dedicated Hardwired IP Cores in Heterogeneous MPSoCs Designed with ESPAM ", *EURASIP Journal on Embedded Systems*, num. Article ID 726096, Vol. 2008, 15 Pages, 2008.

[32] OCP INTERNATIONAL PARTNERSHIP, "http://www.ocpip.org ", 2012

[33] THOMAS M. PARKS AND JOSÉ LUIS PINO AND EDWARD A. LEE "A comparison of Synchronous and Cyclo-Static Dataflow " *Proceeding of the IEEE Asilomar conference on Signal, Systems, and Computers*,oct,1995

[34] P. QUINTON AND T. RISSET "Structured scheduling of recurrence equations : theory and practice ", *Springer verlag, SAMOS*, Sep 2001.

[35] P. QUINTON AND A. M. CHANA AND S. DERRIEN "Efficient Hardware Implementation of Data-Flow Parallel Embedded Systems " *IEEE Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII)*, pages 364-371, 2012

[36] R. SALEH, S. WILTON, S. MIRABBASI, A. HU, M. GREENSTREET, G. LEMIEUX, P. P. PANDE, C. GRECU, A. IVANOV "System-on-chip : Reuse and Integration " *Proceedings of IEEE* num. Vol . 94, 2006.

[37] O. SENTIEYS, J. P. DIGUET, AND J. L. PHILIPPE " GAUT: A High Level Synthesis Tool Dedicated to Real Time Signal Processing Application ", *In European Design Automation Conference, University booth stand*, Sep 2000.

[38] M. SINGH AND M. THEOBALD " Generalized Latency-Insensitive Systems for single-clock and Multi-clock Architecture ", *IEEE Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, 2004.

[39] M. SINGH AND A. AGIWAL "Multi-Clock Latency-Insensitive Architecture and Wrapper synthesis ", *Electronic Notes in Theoretical Computer Science*, 2006.

[40] I. SMARANDACHE, P. LE GUERNIC, "Affine transformations in SIGNAL and their application in the spécification and validation of real-time guif, *Springer-Verlag* num. LNCS 1231, May, 1997.

[41] T. STEFANOV AND C. ZISSULESCU AND A. TURJAN AND B. KIENHUIS, AND E. DEPRETTERE " System Design Using Kahn Process Networks: The Compaan/Laura Approach ", *In DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe, page 10340, Washington, DC, USA, ,* 2004.

[42] W. THIES, M. KARCZMAREK, S. AMARASINGHE "StreamIt : A language for streaming Applications ", *In 11th conf. Compiled Construction*, num. LNCS 2304, pages 179-196. Springer-Verlag, 2002.

[43] XILINX ISE "http://www.xilinx.com, Technical report ", *Xilinx Integrated Software Environement(ISE)*,2012