Military Technical College
Kobry Elkobbah,
Cairo, Egypt

ICEENG 99

2nd International Conference on
Electrical Engineering
ICEENG 99

A HIGH-LEVEL SYNTHESIS METHODOLOGY FOR DEDICATED DSP ARCHITECTURES

E. A. TALKHAN[*] , ALY E. SALAMA[*], AND F. HASHIM[**]

ABSTRACT

In this work we present a proposed High-Level Synthesis (HLS) methodology for dedicated Digital Signal Processing (DSP) architectures. Starting from a purely behavior description of a DSP algorithm, the HLS subtasks namely: the Scheduling, the Allocation, and the Binding are performed to generate an optimized Register Transfer Level (RTL) data path structure which implements the intended behavior while satisfying the timing constraints. The Scheduling and the Allocation subtasks are solved simultaneously in terms of an Integer Linear Programming (ILP) feasibility model. The Binding subtask is solved using a Weighted Bipartite Matching (WBM) algorithm. A 4-point FIR filter is used to demonstrate our methodology in a step wise fashion, from the initially specified behavior to the finally synthesized structure. Simulation results have proved that the finally synthesized data path is truly implementing the initially specified behavior and satisfying the timing constraints.

KEYWORDS: high-level synthesis, digital signal processing, register transfer level, time-constrained optimization, weighted assignment, hardware description language.

---

* Professor, Faculty of Engineering, Cairo University, Giza, Egypt.
** Egyptian Armed Forces.

1. INTRODUCTION

High-Level Synthesis (HLS) bridges the gap between the behavioral description of a circuit and its Register Transfer Level (RTL) structure. It starts with the behavioral description of the circuit, and a set of constraints and goals to be satisfied, and ends with the structure that implements the circuit's behavior while satisfying the goals and constraints. HLS consists of a front-end that translates the input behavior description into an intermediate representation, usually a graph-based, from which the back-end generates the target architecture [1]. The intermediate graph-based representation explicitly shows the control and data dependencies within the input description in the form of a Control and Data Flow Graph (CDFG). In the CDFG, each node represents an operation and each edge represents a data and/or control transfer. The back-end, which is the *core* of the HLS, and the *focus* of this work, involves mainly the following three subtasks:

Scheduling: determines the sequence in which the operations of the CDFG can be executed, by assigning the operations to consective time intervals so called control states or clock cycles. It also determines how many states are necessary to implement a CDFG, and drives utilization figures for functional, storage, and interconnection units.

Allocation: determines the number of each type of the RTL components to be used in the design. For every operation in the CDFG, we need a functional unit that is capable to execute the operation. For every variable that is used across several control states in the scheduled CDFG, we need a storage unit to hold the data values during the variable's life or access time, Finally, for every data transfer in the CDFG, we need a set of interconnection units to realize the data transfer.

Binding: after all operations have been scheduled, and the functional, the storage, and the interconnection units have been allocated. Binding assigns the operations, the variables, and the data transfers, from the behavioral description, into specified instances of the allocated functional, storage, and interconnection units respectively. When multiple alternatives exist for assignment, binding algorithm selects the one that minimally increases the data path cost.

Due to the complexity of the HLS problem and the interdependence among its main subtasks, different methodologies have emerged with different schemes and objectives. The objective of producing suboptimal or globally optimal solutions defines the criteria for classifying these synthesis approaches into heuristic or exact methodologies. The techniques used to deal with the tight interdependence among the HLS subtasks define the criteria for classifying these schemes into independent, iterative, and simultaneous schemes [2]. Since the DSP application domain is resource dominated, area and delay are determined mainly by resource usage and not by a particular binding. Thus we have chosen to solve the Binding (B) subtask, which in this case serves only the purpose of refining the structure information for connectivity, separately after simultaneously solving the Scheduling and the Allocation (S&A) subtasks. In this work we present the formulation of the HLS problem, and expose the essential mathematical and graph theoretical techniques that we need to optimally perform the subtasks of the *back-end* of the HLS system. We also demonstrate our methodology and verify our synthesis results .

2. PROBLEM FORMULATION

Considering our DSP application domain, which is mainly time-constrained, the HLS problem formulation is defined as follows:

Problem 1: *high-level synthesis*
Given:

- *A computational (CDFG) graph describing a digital signal processor behavior.*
- *A time-constraint on the throughput rate and/or the overall execution time of the computation graph on the synthesized architecture.*
- *An RTL module library.*

Find:

- *A minimum cost RTL structure in terms of area, which*
- *Implements the given behavior, and*
- *Satisfies the time constraints.*

According to the human designer's thoughts in searching for an optimal design, our solution methodology is logically partitioned into two phases : a Design Space Exploration (DSE) phase, and a Data Path Construction (DPC) phase. These phases will be explained in some details in the following two sections.

3. DESIGN SPACE EXPLORATION PHASE

In the DSE phase, our design space is explored in the following order : firstly by considering different architectural styles, including different types of resources, different clock cycle lengths, and different topology considerations to satisfy the timing constraints. Secondly by calculating accurate lower and upper bounds on the amount of all the required hardware resources to delimit the search space. Thirdly by solving the Time and Resource Constrained (TRC) optimization problem in terms of an Integer Linear Programming (ILP) feasibility model, where the lower bounds are firstly used to account for the resource allocation constraints then resource counts are incremented until a feasible schedule which uses the minimum amount of resources (optimal schedule) is obtained. The procedures which are involved in the DSE are summarized as follows:

Algorithm 1: *Design Space Exploration*

```
Select: an appropriate hardware module set from the RTL library.
Read : CDFG, parameters of the selected hardware modules, and the time
    constraints.
Determine: the optimal set of clock cycles.
Experiment : with different clock cycle lengths within the optimal set to
        obtain the least critical path length ($T_{longest}$)
If: $T_{longest} > T_{max}$
Then: Apply optimizing transformation, or
      Restart the design exploration phase with faster hardware modules.
Else: Calculate lower and upper bounds on all hardware resources, and
      Solve the TRC feasibility model:
```

```
    Is there a S&A solution to satisfy the following set of constraints?
    Assignment constraints
    Precedence constraints
    Resources allocation constraints
    Time constraints
        Nonfeasible: Increment the number of the heavily utilized
                     resources, and Restart the feasibility model again
        Feasible: Start the data path construction phase
End algorithm 1
```

3.1 Determining a Target Architecture Style

Since HLS implies a huge design space to search, then restricting the HLS towards one target, but flexible architecture style, will tighten the design space and make the high-level synthesis problem more feasible [3]. Thus we have utilized the Finite State Machine with a Data Path (FSMD) model [4], as our generic target architecture. The FSMD implementation consists of a Finite State Machine (FSM) called the control unit, and a data path. The style of the data path, which is our *concern* in this work, is determined in terms of its primitive modules, clocking requirement, and its topology structure as follows:

Modules selection: Since the number of operation types in the DSP application domain is limited. In our methodology , we leave the selection of functional unit types to the user or the expert system. For every type of operation in the CDFG, a list of the available functional unit types is produced to perform such operation. Then the user or the expert system browses and selects at least one from every list for every type covering all the operations types within the CDFG.

Clock cycle determination: Some of the values which contribute in determining the clock period will not be available until at least the floorplanning is done. It is difficult, if not impossible, to assess the influence of these factors during HLS . However, the designer must specify a nominal clock period (or at least, the data path component of the clock period) before scheduling, realizing that the actual clock period will be longer [5]. In our methodology we determine an optimal length for the data path component of the clock cycle such that the slack time within each clock cycle is minimized, as listed below:

```
Algorithm 1.1: clock cycle length determination
    For the selected RTL module set :
      Find the minimum and maximum values of the clock cycle length such that :
        $C_{min}$ = minimum data transfer delay
        $C_{max}$ = register-to-register execution delay of the slowest functional unit type.
      For each selected module type :
        Find the set of clock cycle values that integrally divide the module's register
        to register execution delay, and not less than $C_{min}$
```

From the previous set of clock cycle values, find the value that produces the shortest critical path length of the CDFG, using the unconstrained As Soon As Possible (ASAP) scheduling algorithm [6].

For all module types Find the global value of the clock cycle that corresponds to the minimum value of the critical path length : $C_{global} = C_{optimum}$

End algorithm 1.1.

Topology considerations: Our system can support a synthesis of data path architecture in both: random topology, which is mux-based structure, and linear topology, which is bus-based structure. In random topology architectures, the interconnections between registers and functional units are made of point to point direct links with multiplexers introduced at appropriate places. A linear topology architecture, in contrast, employs a number of buses to implement the interconnections. Consequently we have chosen the linear topology style, due to its regularity, to implement our target data path. Our target data path comprises three major components: functional, storage, and interconnection units, as shown in Fig. 1.

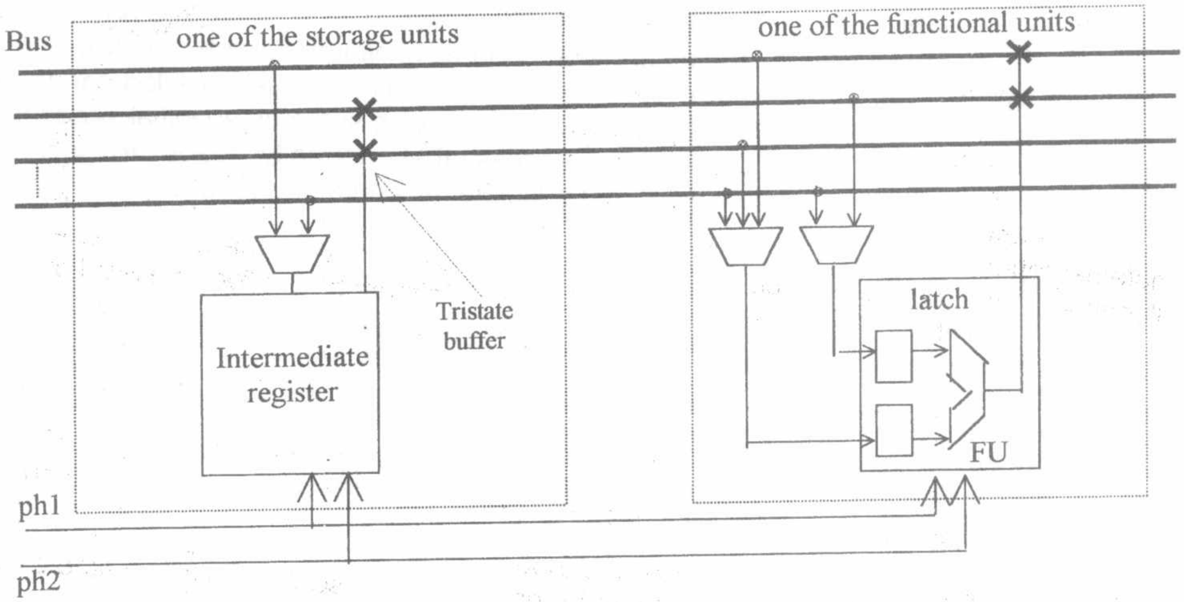


Fig. 1. The style of our target data path.

3.2 Calculating Resource Bounds

In order to calculate resource bounds for all the hardware resources, a storage and a connectivity graphs, are induced from the computational CDFG. In the CDFG, $G\ (N\ ,\ E)$, each node $n_i \,\varepsilon\, N$ represents a code operation, and each edge $e_{ij} \,\varepsilon\, E$ represents a data or control dependency between the operations. In the connectivity graph $G_c\ \ (N_c, E_c)$ every node represents an edge in the corresponding CDFG. An edge $e_c \varepsilon\ E_c$ is defined in G_c when there exists a precedence between two interconnection. These precedences can be derived directly from the CDFG. For our hardware model, the storage graph $G_s(N_s, E_s)$, is actually identical in structure to the interconnect graph. In the storage graph each node $n_s \varepsilon N_s$ corresponds to a variable. The slack time of the node is set to the maximum lifetime of the variable, which is equal to $(t_{alap}(dest) - t_{asap}(source))$, with $t_{alap}(dest)$: the As Late As Possible (ALAP) scheduling time of the variable's destination node, and $t_{asap}(source)$: the As Soon As Possible (ASAP) scheduling time of the variable's source node.

Two operations, variables, or connections can be executed simultaneously when no precedence relationship exists between them, or in other words, when no edge exists between them in the corresponding graph. Then the maximum possible concurrency is obviously equal to the maximum set of nodes without precedence relations or equivalently, the maximum independent set. Thus maximum bounds on all hardware resources can be transformed into the maximum independent set problem which can be solved, for a class of graphs called comparability graphs, in polynomial time (O(N^3)) using the Minimum Flow algorithm [7].

Far more important than the maximum implementation bounds on the hardware resources, are the minimum implementation bounds. Minimum bounds (theoretically optimal) allows us to estimate the absolute minimum area needed for the implementation of a given computational graph. It can also serve as an initial seed for allocation and design space search process that normally leads to faster convergence. Within our frame work, the absolute minimum bound for an execution unit of type r (r = 1 .... R) is defined as : $min_r^{abs} = \left\lceil \frac{n_r \cdot t_r}{T_{max}} \right\rceil$

where n_r is the number of operations to be executed on resource r, t_r is the number of clock cycles it takes to execute an operation on r, T_{max} is the available time given in terms of clock cycles, and $\lceil \; \rceil$ means rounded up to the nearest integer value. The minimum bounds calculation problem for each type of resources is formulated as follows:

Problem 2: *minimum bounds calculation*

Given:

- *A computational graph consisting of n_r identical operations with integer ASAP and ALAP times*
- *The execution time of the operation on the selected hardware resource t_r*
- *The available time constraint T_{max}*

Find:

- *The minimum number of resources min_r needed to complete the operations within the available time.*

This minimum bounds calculation problem cannot be solved directly, but can be defined as an iterative version of its dual formulation as follows:

Problem 2.1: dual problem
Given:

- *A computational graph consisting of n_r identical operations with integer ASAP and ALAP times.*
- *The execution time of the operation on the selected hardware resource t_r*
- *The number of the available resources min_r*

Find:

- *The minimum execution time T_{min}.*

The solution of dual problem (problem 2.1) can be solved in polynomial time using the Earliest Deadline List Scheduling algorithm [8].

Given the solution for its dual problem, the original problem (problem 2) can be solved iteratively as follows:

Algorithm 2: *minimum bounds calculation*

```
Set $min_r$ to the absolute lower bound $min_r^{abs}$
*Given $min_r$:
    Solve the dual problem to determine the minimum execution time $T_{min}$
If $T_{min} > T_{max}$,    or if no solution
    Increment $min_r$, and go to *
Else $min_r$ = minimum bound.
End algorithm 2.
```

All the above described techniques can be applied in an approximately identical fashion for the estimation of the lower bounds of buses and registers. The only difference is that the formulation will consider the interconnect and storage graphs instead of the computational graph.

3.3 Solving the TRC Optimization Problem

After calculating lower and upper bounds on all the required hardware resources, we use the lower bounds to convert the TC optimization problem into an easy to solve Time and Resource-Constrained (TRC) decision problem with a smaller search space. The TRC decision problem is formulated in terms of an Integer Linear Programming (ILP) feasibility model which satisfies both the time and the resources constraints. The notation used in our formulation are defined as follows:

- o_i : a code operation, $i = 1,2,3.........O$, where O is the total number of operations in a CDFG.
- t_i : a schedule time or control step, $i = 1,2,3.........T$, where T is the schedule length or the over all computation time constraint.
- f_i a functional unit, $i = 1,2,3......\ \Sigma_\tau F_\tau$, where F_τ is the number of functional units of type τ, and $\tau = \{ +,*, \text{alu},....\}$.
- $o_i \ \varepsilon \ f_\tau$: indicates that an operation o_i can be mapped to a functional unit of type τ.
- $o_i \rightarrow o_j$: represents an edge or a partial order between operations o_i , o_j, implies that o_i must be executed before o_j.
- x_{o_i, t_i} : a binary variable, its value equals 1 to represent the assignment of a code operation o_i to a time step t_i, and equals 0 otherwise.

The feasibility model does not ask for an optimum, but asks whether a feasible solution exists. Then the feasible solution that uses the least amount of resources will obviously be the optimum one. Thus our formulation includes no objective function but does have the following set of constraints:

Assignment constraint:
The operation assignment constraint ensures that each operation o_i will be assigned to one and only one time step t_i .

$$\Sigma_{t_i \varepsilon \mu(o_i)} \ x_{o_i, t_i} = 1 \qquad \forall o_i \tag{1}$$

Precedence constraint:
The precedence constraint prevents an operation o_j from being scheduled before operation o_i whenever there is a partial order between them such that $o_i \rightarrow o_j$.

$$\Sigma_{\substack{t_i \geq t \\ t_i \ \varepsilon \ \mu(o_i)}} x_{o_i, t_i} + \Sigma_{\substack{t_j \leq t \\ t_j \ \varepsilon \ \mu(o_j)}} x_{o_j, t_j} \leq 1$$

$$\forall t \ \varepsilon \ \mu(o_i) \cap \mu(o_j) \tag{2}$$

Functional units allocation constraint:
The functional unit allocation constraint ensures that no more than F_τ functional units of type τ will be required in the solution.

$$\sum_{\substack{o_i \,\varepsilon\, f_\tau \\ t_i \,\varepsilon\, \mu(o_i)}} x_{o_i,t_i} \le F_\tau \qquad \forall\, \tau, t \tag{3}$$

Register allocation constraint:

The register allocation constraint ensures that there are no more than R variables whose lifetimes overlap while crossing any time step. For each time step, we ignore possible scheduling of producer-consumer (o_i, o_j) pairs that completely lie above or below the current time step. we accomplish this by canceling out possible schedule of both producer and consumer in one side of the current time step as follows: Giving every possible schedule of the producer (consumer) above (below) the current time step, $t_{current}$, a positive sign. Giving every possible schedule of the produce (consumer) below (above) the current time step, $t_{current}$, a negative sign. Thus the register allocation constraint calculates twice the number of cross edges $(o_i \rightarrow o_j)$ at each time step as given below:

$$\sum_{o_i \rightarrow o_j} \Big(\sum_{\substack{t_i \le t \\ t_i \,\varepsilon\, \mu(o_i)}} x_{o_i,t_i} - \sum_{\substack{t_i > t \\ t_i \,\varepsilon\, \mu(o_i)}} x_{o_i,t_i} + \sum_{\substack{t_j > t \\ t_j \,\varepsilon\, \mu(o_j)}} x_{o_j,t_j} - \sum_{\substack{t_j \le t \\ t_j \,\varepsilon\, \mu(o_j)}} x_{o_j,t_j} \Big) \le 2R \qquad \forall\, t \tag{4}$$

Bus allocation constraint:

The bus allocation constraint ensures that at each time step, no more than B buses are required to transfer data between functional units and registers. Data broadcasting is modeled using fixed timing constraints on all pairs of destinations and selecting one of the destination operations to contribute to the bus count. Thus the number of buses required at a time step t_i equals the number of *distinct* input and output variables of all operations assigned to this step.

$$\sum_{\substack{o_i \\ t_i \,\varepsilon\, \mu(o_i)}} \big(in(o_i) + out(o_i) \big)\, x_{o_i,t_i} \le B \qquad \forall t \tag{5}$$

Time Constraint

The time constraint on the overall execution time (T_{max}) is imposed explicitly to prevent scheduling leave operations beyond the time constraint, as follows:

$$\sum_{t_i \varepsilon \mu(o_i)} (t_i + D_i - 1) \quad x_{o_i, t_i} \leq T_{max} \qquad \forall\, o_i \text{ without successors } \mathbf{(6)}$$

4. DATA PATH CONSTRUCTION PHASE

In the DPC phase, the operations, the variables, and the data transfers, from the scheduled CDFG, are bound to a specified instances of the allocated hardware resources. When binding functional units and registers, the objective is to minimize the possible increase in area, in terms of the number of induced multiplexer inputs and tristate buffers. Whereas, when binding buses, the objective is to minimize the data transfer delay time, in terms of the maximum total loading of the data source (functional units or registers) and the maximum total loading of the data carrier (buses) during every phase of the clock cycle.

Operation Binding (OB), Variable Binding (VB), and Data Transfer Binding (DTB) are tightly related to each other. There is no obvious ordering of these subtasks. However, we know that the number of induced interconnections between functional units and buses is determined by both OB and DTB. Also, the number of induced interconnections between registers and buses is determined by both VB and DTB. While the interaction between OB, and VB is indirect. In our methodology, we perform OB, and VB separately, while incorporating means for interconnections minimization in both subtasks. On the other hand, we perform the DTB subtask while evaluating its effect on the data transfer time, the goal is to balance the load among data sources and carriers.

The functional units and the registers binding algorithms start with a scheduled CDFG and ends with a partially synthesized (scheduled and partially bound) data path as listed below:

Algorithm 3: *functional units and registers binding*

```
    Cluster the operations/variables in the scheduled CDFG, such that the
            scheduling/life time of each operation/variable in a cluster overlaps with
            the scheduling/life times of all the operations/variables in the same
            cluster.
    Construct a Weighted Bipartite Graph (WBG) from the set of the allocated
            functional units/registers, and the members of each cluster
    Assign the operations/variables of the first cluster to the allocated number of
           functional units/registers simultaneously.
    While assigning the remaining clusters, ensures that :
      An operation/variable can be assigned to a functional unit/register only if its
      schedule/life time does not overlap with the schedule/life times of all
    the operations/variable which were already assigned to that functional
      unit/register.
    If more than one operation/variable is considered for binding, the one that
       induces the least increase in interconnection cost is chosen.
End algorithm 3.
```

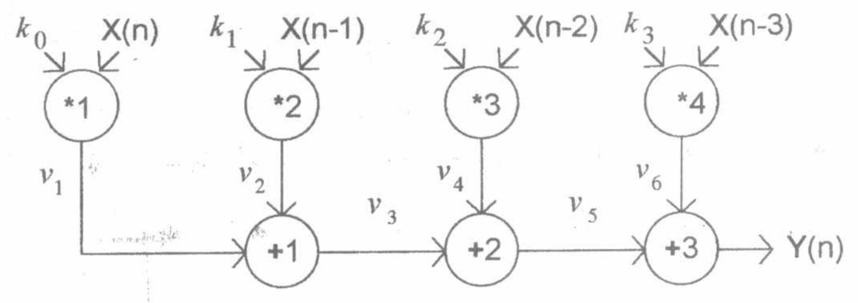
The process for data transfers to buses binding becomes clear after both the functional units and the registers bindings have been performed. The data transfers within the partially bound data path are assigned to the allocated number of buses as follows:

Algorithm 4: *data transfers to buses binding*

```
Cluster the transfers in each phase j of each clock cycle i as a set $T_{ij}$.
    For each transfer set $T_{ij}$ :
      Construct a WBG from the set of the allocated buses B, and the members
      of the set $T_{ij}$.
      Assign the data transfer of the first cluster to the allocated number of buses
      simultaneously.
      While assigning the remaining clusters, ensures that :
         data transfer can be assigned to a bus only if its schedule time does not
         overlap with the schedule times of all the data transfers which were
         already assigned to that bus.
      If more than one data transfer is considered for binding, the one that
      induces the least increase in the data transfer delay time is chosen.
End algorithm 4
```

5. SYNTHESIS RESULTS

The 4-point FIR filter is used to demonstrate our methodology, and to verify our synthesis results. The behavior description of the 4-point FIR filter is given in the form of its CDFG and a time constraint on its data introduction interval (T_{sample} = 300 ns), as shown in Fig. 2.



T_{sample} = 300 ns, Y(n) = k_0 X(n) + k_1 X(n-1) + k_2 X(n-2) + k_3 X(n-3)

Fig. 2 The behavior description of the 4-point FIR filter

Assume that we have manually selected a multiplier of 80 ns, an adder of 40 ns, and a latch of 20 ns delay times. Then according to Algorithm 1.1, the clock cycle length will be 20 ns which results in a 280 ns critical path length. Since the resulting critical path length is less than the data introduction interval, no optimizing transformation will be needed, and the style of the target data path will be non-pipelined.

In order to reduce the search space and to convert the given TC optimization problem into TRC decision problem , maximum bounds of 4 multipliers, 1 adder, 4 registers, and 8 buses, and minimum bounds of 3 multipliers, 1 adder, 1 register, and 2 buses are produced using the Minimum Flow algorithm and Algorithm 2 respectively.

The LP feasibility model is then constructed using both the time constraint and the minimum bounds on the resources. It includes 23 binary variables which are required to satisfy 66 constraints. The linear programming package of the Mathematica Environment [9] is used to solve the LP problem. As a result we have obtained an optimal Scheduling and Allocation (S&A) solution with 3 multipliers, 1 adders, 2 registers, and 2 buses.

Operations to functional units and variables to registers bindings are obtained using Algorithm 3. As a result a partially synthesized data path, is obtained with 15 data transfers. These transfers are bound to the allocated 2 buses using Algorithm 4 to obtain the finally synthesized data path of the 4-point FIR filter, shown Fig. 3, with 8 tristate buffers and 11 multiplexer inputs. The data transfers are evenly distributed among the buses, which results in a maximum load of functional units and registers equal to 1 tristate buffer, and a maximum load of bus equal to 6 multiplexer inputs.

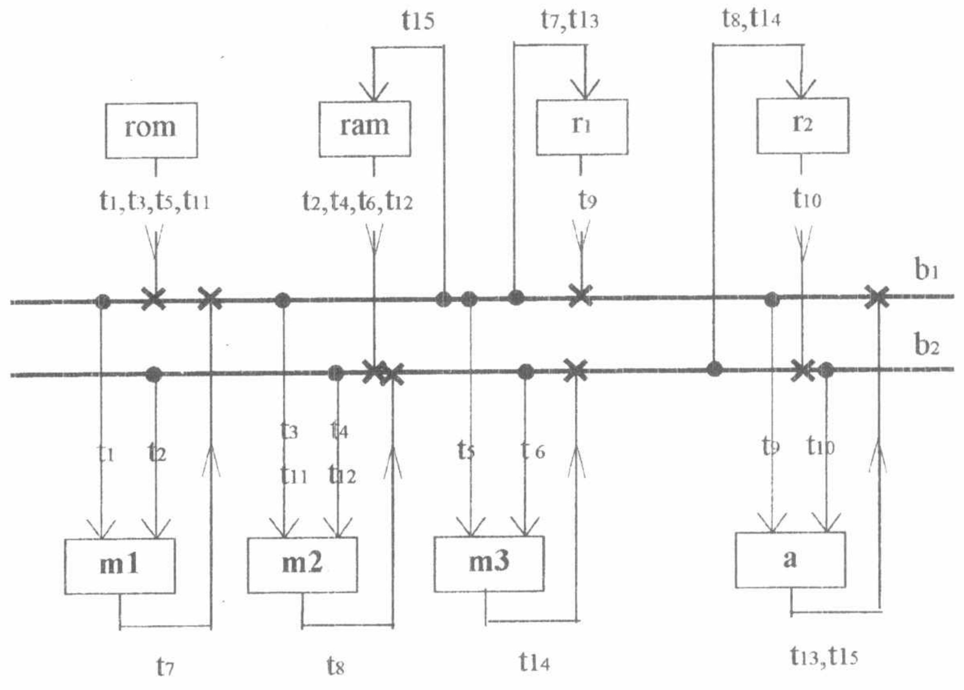


Fig. 3 The finally synthesized data path .

The IEEE standard Hardware Description Language (VHDL) is used to model both the behavior and the finally synthesized data path structure [10]. Then both models are simulated using the VHDL Compiler-Simulator Environment [11]. Simulation results, shown in Fig. 4, verify that: using a 20 ns clock cycle length, the input data are up updated every 15 clock cycles and the expected output is produced every 15 clock cycles as well, which indicates a complete coincidence between the performance of the initially specified behavior and the finally synthesized structure.

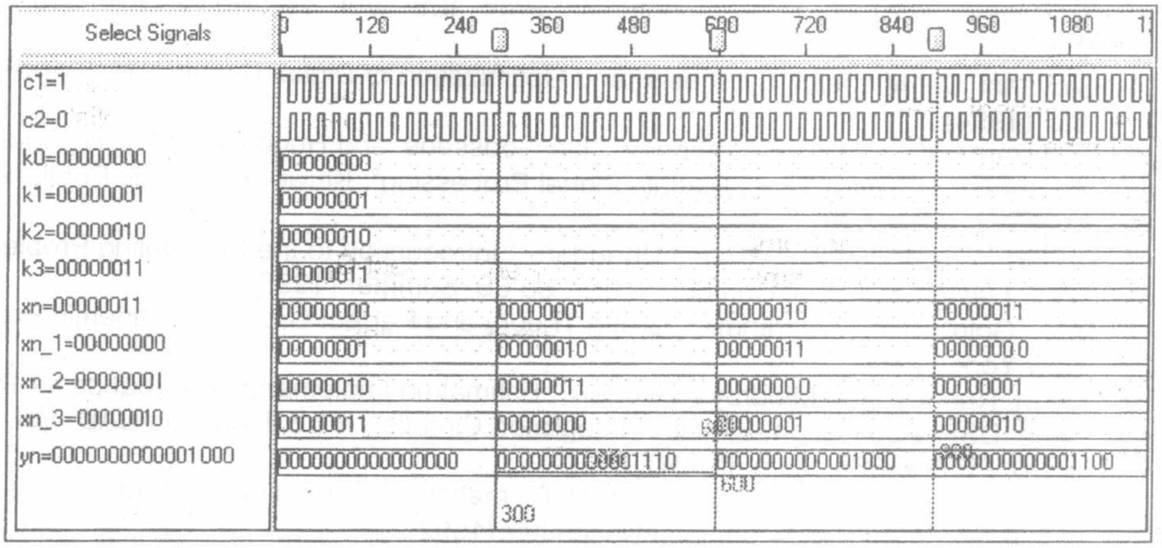


Fig. 4. Simulation results of the finally synthesized data path

6. CONCLUSION

The proposed HLS methodology utilizes a Finite State Machine with a Data Path (FSMD) as a generic target architecture, which uses a bus-based topology for the data path and a two-phase clocking scheme for the control path. It incorporates two phases: a Design Space Exploration (DSE) phase and a Data Path Construction (DPC) phase.

In the DSE phase, the Scheduling and the Allocation (S&A) subtasks are solved simultaneously, to overcome their tight interaction, which results in an optimal scheduling that uses the minimum amount of hardware resources and satisfies the timing constraints. In the DPC phase, the allocated hardware resources are connected together to construct the data path such that the intended behavior is implemented, the possible increase in area is minimized, and the time constraints are preserved.

The demonstrating example has showed that how an optimized data path can be synthesized from a purely behavior description. Then simulation results have proved that the finally synthesized data path is truly implementing the initially specified behavior and satisfying the timing constraints.

REFERENCES

1- Michael C. McFarland, Alice C. Parker, and Raul Camposano, "The High-Level Synthesis of Digital Systems", Proc. IEEE, Vol.78, No.2, PP. 301-318, February (1990).

2- D.D. Gajski, N. Dutt, A. Wu, and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design", Kluwer Academic (1992).

3- Hugo De Man, Fracnky Catthoor, Gert Goossens, Jan Vanhoof, Jef Van Meerbergen, Stefaan Note, and Jos Huisken, "Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms", Proc. IEE, Vol. 78, No. 2, PP. 319-334, February (1990).

4- Giovanni De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, Inc. (1994).

5- Jan Vanhoof, Karl Van Rompaey, Ivo Goossens, and Hugo De Man, " High-Level Synthesis for Real-Time Digital Signal Processing", Kluwer Academic Publishers, (1993).

6- Robert A. Walker, and Samit Chauduri, "Introduction to the Scheduling Problem", IEEE Design & Test of Computers, PP. 60-69, Summer (1995).

7- M.C. Golulmbic, "Algorithmic Graph Theory and Perfect Graphs", Academic Press, New York, (1980).

8- Jan M. Rabaey, and Miodray Potkonjak, "Estimating Implementation Bounds for Real Time DSP Application Specific Integrated Circuits", IEEE Trans. on CAD, Vol. 13, No. 6, PP 669-683, June (1994).

9- " Mathematica: A System for Doing Mathematics by Computer", Version 2.2, User's Guide, Wolfram Research, Inc., December (1992).

10- Fawzy Hashim, " High-level Synthesis of Dedicated DSP architectures', Ph.D. Thesis, Faculty of Engineering, Cairo University, Egypt (1998).

11- Accolade Design Automation's PeakVHDLTM Simulator Demonstration, Version 1.14, Accolade Design Automation, Inc., (1998).