EasyChair Preprint

№ 2257

# Determine the Interconnection of a Hardware Implementation for DSP Applications

Jehad Ahmad Ghanim and Ali Shatanawi



November 13, 2020

# Determine the Interconnection of a Hardware Implementation for DSP Applications

*Jehad Ahmad Ghanim*
*Department of Computer Science*
*AlKharj Community College/*
*Sattam bin AbdulAziz University*
*AlKharj, Saudi Arabia*
*j.ghanim@psau.edu.sa*

*Ali Shatnawi*
*Department of Computer Engineering*
*Jordan University of Science and Technology*
*Box 3030, Irbid 22110, JORDAN*
*ali@just.edu.jo*

**Abstract - In VLSI design the hardware is implemented with some objective and constrain functions (as lower number of hardware used). When the system contains a lot of processing elements (PEs) and memory registers, the cost of the interconnections becomes of great issue and must be minimized. The work in the field of determination of the interconnection for a hardware implementation is not very common. In high-level synthesis it is usually considered the time scheduling and processor assignment from a given DFG. However, the cost of interconnection is not widely discussed and is left to a hardware system to determine it. In this paper, a technique for determining the interconnection in a hardware design is proposed. The objective function is the minimum number if hardware used and the constrain is minimum iteration period bound. This interconnection is shown to accomplish cost optimality in terms of minimizing the number of multiplexers used.**

.

***Keywords: Datapath, DFG, high-level synthesis, multiplexers***

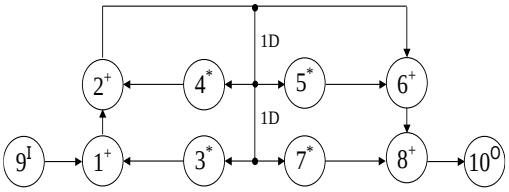
## I. Introduction

Digital signal processing (DSP), communications, and image processing are computationally intensive, and thus demand systems with high computational power. The high computational power is necessary to accomplish the given tasks in real time. Due to the inherent parallelism of DSP applications, a multiprocessor system is a natural choice for the implementation of these applications [1].

.

*The technique used* to design digital systems at the high level is typically referred to as the high-level synthesis (HLS). This technique is used to produce efficient hardware implementations for the given tasks. The HLS process starts with the specification of the algorithmic level behavior (input-output specifications) of a given digital system, and completes by finding the data path realizing the given I/O specifications. The algorithmic level behavior is usually represented by a data flow graph (DFG). An example of a DFG is depicted in Fig 1. Through the HLS procedure objective functions and constraints must be met.

In this paper, a technique [12] to determine the interconnection of a given hardware implementation is proposed. The result of this technique is a schedule matrix used to represent a data path of heterogeneous multiprocessor system's implementation.




+:Adder of 1 computational delay
*:Pipelined multiplier of 2 computational delays

Fig 1: DFG of the second order IIR filter

## II. Previous Work

Synthesis tools that are used to generate valid implementations for digital systems are present since the 1970s. Many of them are targeted for the generation of data paths of general-purpose digital systems.

Some of the synthesis tools are specialized in DSP applications. LAGER is a data path compiler that is specialized in DSP applications

[3]. The algorithmic behavior specification is expressed in an Assembly-like language program. The main disadvantage of this synthesis tool is that, the multiprocessing system is limited to four processors communicating by serial interfaces. MARS (Minnesota Architecture Synthesis) [8] does not consider the minimization of the interconnection network. Furthermore, MARS execution time is unpredictable as the course of action of its procedure tremendously varies with the graph topology [9].

## IV. Determining the Interconnection

Since there are a number of processing elements (PEs) and registers which are used to handle data tokens that are produced by other PEs or stored in registers, multiplexers are required to control the data transfer between these PEs and the registers. Each PE or register having more than source of data tokens requires a multiplexer to select between these sources. The selection inputs of each multiplexer must be set to select the value of the required source at the different control steps. The multiplexers and the write-enable signals from the control unit assure the proper operation of the system (according to the I/O behavioral specifications).

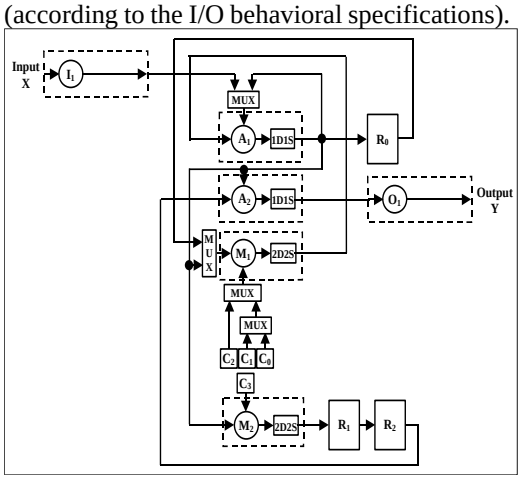



Fig 2: Hardware implementation for the second-order IIR filter shown in Fig 1

The interconnection minimization is done by reducing the size of each multiplexing unit. This minimization is performed by decreasing the number of sources of the variables that are attached to the inputs of each multiplexer.

There are two groups of multiplexers, the PEs multiplexers and the registers multiplexers. The sources of the PE-multiplexers are any PE, register, or constant-register. The sources of the register-multiplexer are any PE or register. Since each PE has two inputs, each require a distinct group of input multiplexers. In this case, there is a possibility that a source of data operands may be attached to the inputs of each multiplexer group. Thus, if it is possible to attach each source of data operands to only one multiplexers group, the total size of the multiplexers is reduced. The lower bound on the size of the multiplexers groups of any PE is determined by the total number of different sources of this PE. The algorithm that attempts to minimize the size of both multiplexers' groups of any PE (and thus the number of multiplexers) is stated below:

1. Put all PEs in a non-selected PEs list
2. Select a PE from this list and set it as a target PE. This PE is removed from the non-selected PEs list.
3. Group all the sources (PEs and registers) of the data tokens for the target PE in a sources-list. This list also contains the control steps at which each source is accessed, since each source may occur several times in this list.
4. Calculate the number of occurrences of each source in this list. This number equals to the number of different references to a source in an iteration period.
5. Sort the sources in the sources-list according to the number of occurrences of each source in descending order
6. Take the first source (the one with the largest number of accesses found) and remove it from the sources-list
7. Assign each access of the taken source to the entry of the required control step in the row of the multiplexers group of the target PE that has the highest number of free cells (in the PEs multiplexers source selection matrix). If the entry of the required control step is not free, assign it to the entry of the required control step in the row of the other multiplexers group.
8. If the source-list is not empty go to step 6
9. If the non-selected PEs list is not empty go to step 2

The register-multiplexer minimization process described in the register allocation algorithm in the previous section is done in case that a variable consists of two elements only. In this case, the two elements are assigned to the same register if possible. Thus, element produced in the previous iteration is not required to be transferred between multiple registers, since this element is already stored in a register.

Table 1 and table 2 show the source of the data for each PE and register of the example (Fig 1) at each control step. From these tables, the values of the select lines of the multiplexers of each PE and register at each control step are determined.

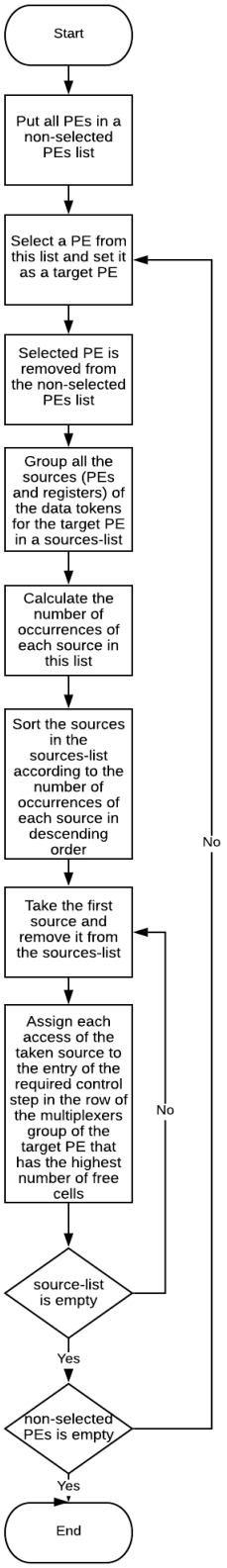



Fig 3. Multiplexers minimization Algorithm

The columns determine the control steps (except the last one) and the rows determine the PEs. Each cell in the table determines the type of the source of the variable and its number. The letters A, M, R, and C denote adders, pipelined multipliers, registers, and registers that store constants (multiplier constants) respectively. The number next to each letter identifies the source from the group of sources of the same type. The last column determines the number of inputs of the multiplexer, or the number of different sources that are connected to the corresponding multiplexer. The row with dashed entries in the multiplexer number (MUX#) or register number (register#) columns means that it does not have any multiplexers or registers.

| Processor \ Control Step | MUX# | 0 | 1 | 2 | # |
|---|---|---|---|---|---|
| A1 | MUX1 | M1 | M1 | M1 | 1 |
| | MUX2 | I1 | A1 | A1 | 2 |
| A2 | MUX1 | R2 | - | - | 1 |
| | MUX2 | A1 | - | - | 1 |
| M1 | MUX1 | R0 | R0 | A1 | 2 |
| | MUX2 | C0 | C1 | C2 | 3 |
| M2 | MUX1 | - | - | A1 | 1 |
| | MUX2 | - | - | C3 | 1 |
| I1 | - | - | - | - | 0 |
| O1 | - | - | A2 | - | 1 |

Table 1: The PEs multiplexers source selection matrix

| Register \ Control Step | 0 | 1 | 2 | # |
|---|---|---|---|---|
| R0 | - | - | A1 | 1 |
| R1 | - | M2 | - | 1 |
| R2 | R1 | - | - | 1 |

Table 2: The registers multiplexers source selection matrix

In table 1, since each source appears only in one multiplexer row for each PE, hence, the bound on the size of each multiplexer is achieved.

Due to the repeated nature of DSP algorithms, the sequences in the source selection matrices are repeated every iteration. Thus, the iteration period imposes an upper bound on the number of inputs for each multiplexer. Number of inputs must be as $No\_Inputs \leq T$.

The dashed entries in the source selection matrices of the PEs and registers represent the case of no input and thus the number-of-inputs for each multiplexer can be reduced by the number of these dashed entries. In addition, many different variables can be taken from the same source (PE or register).

All entries in the source selection matrices that belong to the same source must be

set to the same input line of the multiplexer. Furthermore, because the registers can latch the data based on a write-enable signal from the control unit, consecutive elements of a variable that are stored in the same register do not require additional inputs. This is because the previous element is still in the register until its lifetime ends and the next element is latched by a write-enable signal from the control unit. For example, in register number 0, the source of the third entry is the first add PE, but the source of the first entry is the register 0 itself. Since the element is already in register 0, this entry does not need an input for the multiplexer of register number 0. Thus, the minimum number of inputs (No_Inputs) for each multiplexer is limited by the maximum number of different sources of each variable.

| Filter Type | Number of MUXs without algorithm | Number of MUXs with algorithms |
|---|---|---|
| Second-order IIR filter | 8 | 4 |
| fifth-order wave digital elliptic filter | 36 | 11 |
| fourth-order Jaumann wave digital filter | 17 | 6 |
| all-pole lattice filter | 16 | 7 |

Table 3. The number of multiplexers required before and after the use of the minimization algorithm

| Filter Type | Central Memory | Distributed Memories |
|---|---|---|
| | number of MUXs | number of MUXs |
| Second-order IIR filter | 1 | 0 |
| Fifth-order wave digital elliptic filter | 4 | 0 |
| fourth-order Jaumann wave digital filter | 2 | 0 |
| all-pole lattice filter | 2 | 0 |

Table 4. The required number of multiplexers for both types of memory systems

Instead of using $(N\times 1)$ multiplexers for the representation of each PE multiplexer group or register-multiplexer, N being the minimum power of 2 larger than (No_Inputs), a group of (No_Inputs-1) two-input multiplexers [11] can be used to form *multiplexing unit*. This is because the number of multiplexer inputs N is usually more than the needed number of inputs (No_Inputs). These (No_Inputs-1) two-input multiplexers are connected as a binary tree to form a multiplexing unit with a lower cost than that of the $(N\times 1)$ multiplexer. Since each two-input multiplexer has only one select line, a group of $\lceil \log_2(\text{No\_Inputs}) \rceil$ select lines are required for the new multiplexing unit. The least significant select line is connected to the leaf multiplexers and the most significant select line is connected to the root multiplexer.

A row in the source selection matrices with only one input (indicated from the last column) implies that the corresponding hardware unit (PE or register) does not require a multiplexer and is connected directly to the source of that input. These rows with zero number of inputs means that the corresponding hardware unit does not take any input from any other hardware units in the system. This is the case for the input virtual PE. This PE takes its input stream from outside the system; thus, it has a zero number of inputs with respect to other hardware units.

From table 1, it is clear that there are two different sources at MUX2 of the first adder and at MUX1 of the second multiplier. Thus, the multiplexers used are $(2\times 1)$ multiplexers. The second PE of each type has only one operation to process, thus it does not need any multiplexer, and it is connected directly to its source of data. Clearly from table 2 the registers do not need any multiplexers.

## III. Conclusion

A new technique to determine the interconnection is presented here. It is used to assign the data sources to the inputs of the processing elements is also presented. This algorithm minimizes the total number of such sources assigned to each input multiplexer. As a result, the total size of multiplexing units is minimized.

## REFERENCES


[1] D. J. DeFatta, J. G. Lucas, and W. S. Hadgkiss, “Digital signal processing, a system design approach,” John Wiley & Sons, 1988.

[2] M. C. McFarland, A. C. Parker, and R. Camposano, “The high-level synthesis of digital systems,” Proceedings of the IEEE, vol. 78, no. 2, pp. 301-318, Feb. 1990.

[3] J. M. Rabaey, S. P. Pope, and R. W. Brodersen, “An integrated automated layout generation system for DSP circuits,” IEEE Trans. Computer-Aided Design, vol. CAD-4, no. 3, pp. 285-296, Jul. 1985.

[4] H. Deman et al., “Cathedral II: A silicon compiler for digital signal processing,” IEEE Design and Test, vol 3, no.6, pp. 13-25, Dec. 1986.

[5] B. S. Haroun, and M. I. Elmasry, “Architectural synthesis for DSP silicon compiler,” IEEE Trans. Computer-Aided Design, vol.8, no. 4, pp. 431-447, Jun. 1990.

[6] C. T. Hwang, J. H. Lee, Y. C. Hsu, and Y. L. Lin "A formal approach to the scheduling problem in high level synthesis," IEEE Trans. Computer-Aided Design, vol. 10, no. 4, pp. 464-475, Apr. 1991.

[7] F. F. Yassa, J. R. Jasica, R. I. Hartley, and S. E. Noujaim, "A silicon compiler for digital signal processing: methodology, implementation, and applications," Proceeding of the IEEE, vol. 75, no. 9, pp. 1272-1282, Sep. 1987.

[8] -Y. Wang, and K. K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, and allocation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 3, pp. 274-295, Mar. 1995.

[9] Shatnawi, "Compile-time scheduling of digital signal processing data flow graphs onto homogeneous multiprocessor systems," Ph.D. Thesis Department of Electrical and Computer Engineer, Concordia University, Montreal Canada, Apr. 1996.

[10] Shatnawi, M. O. Ahmad, and M. N. S. Swamy, "Scheduling of DSP data flow graphs onto multiprocessors for maximum throughput," Paper no.548, The 1999 IEEE International Symposium on Circuits & Systems, May 30-June 2, 1999, Orlando, Florida.

[11] C.-J. Tseng, and D. P. Siewiorek "Automated synthesis of data paths in digital systems," IEEE Transactions on Computer-Aided Design, vol. CAD-5, no. 3, pp. 379-395, Jul. 1986.

[12] J. Ghanim, "High Level Synthesis of Integrated Heterogeneous Multiprocessor Systems for Digital Signal Processing Applications," Msc. Thesis Department of Electrical Engineering, Jordan University of Science and Technology, Irbid Jordan, Jan. 2000.