**evolFIR: Evolving redundancy-free FIR structures**

Szilvia Zvada & Gabriella Kókai  
Department of Programming Systems  
University of Erlangen-Nuremberg  
Martensstr. 3,  
D-91058 Erlangen  
Germany  
{zvada,kokai}@cs.fau.de

Róbert Ványi  
Institute of Informatics  
University of Szeged, Árpad tér 2,  
H-6720 Szeged, Hungary  
robert@vanyi.org

Hans Holm Frühauf  
RF&Microwave Department  
Fraunhofer IIS, Am Wolfsmantel 33,  
D-91058 Erlangen, Germany  
fhf@iis.fraunhofer.de

**Abstract**

Finite impulse response (FIR) structures are the most commonly used digital filters and can be found in various areas of everyday life. In this paper we introduce a novel approach to construct and optimize a redundancy-free design of such filter structures. Relying on attribute grammars and derivation-tree based genetic programming the evolFIR system can restrict the search space to exactly those filter descriptions which fulfill each of the specified constraints. Furthermore, due to the sophisticated tree representation form, the resulting filter descriptions are not only valid, but also free of redundancy.

1 Introduction

Digital filters are the hot-spot of digital signal processing and finite impulse response (FIR) structures [7] especially are in the focus of intensive research. They are an extension of FIR filters, and can deal with enormous data rates and high bandwidth. Evolutionary optimization methods (especially genetic algorithms) have been established for a long time in various stages of the chip design process, though most of them are applied in the stage of physical design to optimize the placing and routing of building blocks [5, 6, 9].

In the project **evolFIR** (cf. Figure 1), we optimize the logic design of FIR structures with higher order and higher degree. Our main objective is to obtain a redundancy-free filter design by reusing single components or larger modules as often as possible. We achieve this by employing a novel evolutionary engine based on derivation tree based genetic programming (DTGP) [10].

The individuals of the evolutionary process are special derivation trees of an attribute grammar. The trees of the grammar define adequate combinations of functional elements (adders or multipliers) and logical primitives (shifts and delays). This means, that each tree exactly describes the requested transfer function. Moreover, trees are enhanced by attributes, that can indicate certain features of the trees. They can also represent hardware-specific optimization constraints (the maximum number of input/output bits of a multiplier), which are generally optimized in subsequent steps of the chip design. During the optimization, the derivation trees are manipulated by evolutionary tree-operators, that are designed to keep the trees valid all the time. Hence, the search space of this complex optimization problem is greatly reduced such that it solely contains the valid filter structures. Finally, the best trees found by the optimizer are translated into a hardware description language as the input for the synthesis and implementation processes.

![Figure 1. The evolFIR system.](image-url)
2 Representing FIR structures by derivation trees

In contrast to the main stream, the evolFIR system (see Figure 1) is a logic design tool that operates on tree-like structures. Our method follows a quite unconventional approach. From a theoretical point of view, the aim of filter optimization is to find a graph topology where nodes represent the basic elements of a block diagram and directed edges indicate the wires connecting those elements. Thus, we employ attribute grammars [4] to generate special derivation-trees that correspond to the allowed FIR structures of the specified filter coefficients.

2.1 The underlying attribute grammar

The rewriting rules of the context-free grammar describe the topology of the basic elements of the block diagrams as follows:

\[
\begin{align*}
\text{EXPR} & \rightarrow \text{add} \ \text{EXPR} \ \text{EXPR} \\
\text{EXPR} & \rightarrow \text{delay} \ \text{EXPR} \ \text{int} \\
\text{EXPR} & \rightarrow \text{shift} \ \text{EXPR} \ \text{int} \\
\text{EXPR} & \rightarrow x \\
\text{EXPR} & \rightarrow \text{int}
\end{align*}
\]

The terminal symbols add, sub, delay, shift and mul refer to the function blocks, while \( x \) is the input data stream. The int generally denotes an arbitrary constant value on some input port. In case of shift and delay, however, the int indicates the number of shifts and delays to be performed. EXPR is a nonterminal symbol and used as a synonym for expressions, that is for the transfer function of the desired filter structure, in our case. On the right hand side of a rule, EXPR denotes the inputs of a particular function block. On the left-hand-side, it indicates the output values.

Obviously, any valid connection of the given function blocks can be generated with these rules. Yet, there is still a vast number of possibilities to combine the basic elements to valid block diagrams. We, however, are only interested in particular block diagrams. Namely, those which describe the current transfer function. Due to the huge search space, the approach of picking block diagrams randomly and hoping they will be accurate is not a viable method. Therefore, in evolFIR we have turned the table and instead of performing an actual validation check a sophisticated random tree generator is employed to create derivation trees representing solely valid block diagrams.

2.2 Describing the transfer function

First, we associate an attribute \( \text{poly} \) with the nonterminal EXPR to describe the function currently represented by EXPR. Formally, the transfer function of a higher order and higher degree FIR structure is described as a sum of unique polynomial component terms:

\[
y[t] = \sum_{k=0}^{n} \alpha_k \cdot 2^{\gamma_k} \cdot \alpha_k^{\delta_k} \cdot \beta_k^{\gamma_k} \cdot \gamma_k^{\beta_k}, \tag{1}
\]

where each \( 0 \leq \delta_k \leq P \), each \( 0 \leq \beta_k \leq M \) and \( n \leq \max(M, P) \). \( P \) is greatest exponent of \( x \), thus it denotes the degree of the FIR structure. The greatest number of delays, \( M \), is referred to as the order of the FIR structure. Hence, we define \( \text{poly} \) as a list of such component terms. The actual value of the \( \text{poly} \) is decisive during the tree generation.

2.3 Tree generation

In evolFIR the derivation trees are generated randomly. The tree generation starts with the root node EXPR of the tree, where the attribute \( \text{poly} \) defines exactly the polynomial transfer function of the desired filter. Thus, we construct our topology backwards, starting from filters output. The function blocks are “added” to our structure by applying a rewriting rule. Whether a rewriting rule is applicable depends on the current value of the attribute \( \text{poly} \), the hardware constraints, and the hardware-specific optimization parameters. For instance in Figure 2, the \( \text{poly} = 5x \) could be gained by addition or multiplication, but not with shift or delay. Therefore, only the rules add and mul are adequate candidates. After the rule has been selected it is applied, i.e. the tree is extended with the corresponding new nodes. Then, the inherited attributes are evaluated and set in the newly generated nodes preparing them for the next rewriting step. The process is continued with the leftmost nonterminal node. When the generation of a subtree is com-
2.4 Decomposing transfer function

In this section the decomposition of the transfer function through the function blocks is explained.

2.4.1 Addition

When we apply the rewriting rule $\text{EXPR} \rightarrow \text{add} \text{EXPR} \text{EXPR}$, then we assume implicitly that the polynomial represented by $\text{poly}$ has been gained by adding two sub-polynomials $\text{poly}_1$ and $\text{poly}_2$. In this case decomposing $\text{poly}$ means randomly find two appropriate summands. We can decompose $\text{poly}$ if and only if it is neither a constant polynomial nor one of the base polynomials $x$ or $-x$. That is:

If this holds, we take one component term $c_i x^{\delta_i} z^{-\beta_i}$ $(0 \leq i \leq n)$ of $\text{poly}$ after the other, and split them in two. Concretely, it means to split up the coefficient $c_i$, since addition has no impact on either the exponents of $x$ or on the amount of delays $z^{-\beta_i}$. Basically, we do this in the following manner:

1. First, we randomly select an integer $\xi$ from the interval $[0, 2^{m}]$, where $2^{m}$ is the smallest power of 2 that is greater than $|c_i|$. The exponent $m$ is also referred as the word-width of $c_i$. Practically, $m$ is gained by taking the number of bits needed to represent $c_i$:

\[
m = \lceil \log_2 (|c_i|) \rceil .
\]

From practical point of view, the choice of $2^m$—i.e. the next power of 2—enables us to construct some favorable decompositions. Such a decomposition is for instance, $7x = 8x - 1x$. Namely, $8x$ can be reduced to $x$ in a single step by using a shift and shifting is generally cheaper than any other operation.

2. Based on this random integer $\xi$, we divide the coefficient $c_i$ into $c_i = c_i^+ + c_i^-$, where

\[
\begin{align*}
c_i^+ &= \xi \quad \text{and} \quad c_i^+ = c_i - \xi , & \text{if } c_i \text{ is positive} \\
c_i^- &= -\xi \quad \text{and} \quad c_i^- = \xi - c_i , & \text{otherwise}.
\end{align*}
\]

Hence, the two sub-polynomials $\text{poly}_1$ and $\text{poly}_2$ are obtained in the following form:

\[
\begin{align*}
\text{poly}_1 &= c_i^+ x^{\delta_i} z^{-\beta_i} + \ldots + c_n^+ x^{\delta_n} z^{-\beta_n}, \\
\text{poly}_2 &= c_i^- x^{\delta_i} z^{-\beta_i} + \ldots + c_n^- x^{\delta_n} z^{-\beta_n}.
\end{align*}
\]

Figure 3. Derivation tree representing a recursive filter

We emphasize some hardware-related constraints and factors that we encountered while decomposing $\text{poly}$. The most fundamental of them are related to the generation of the random split value $\xi$.

Recall that we choose $\xi$ from the interval $[0, 2^{m}]$. Unfortunately, not every value in the interval $[0, 2^{m}]$ is an appropriate choice as we could see in Figure 3. In order to prevent such cases, we have to exclude those values of the interval that would lead to such inconsistency. It is apparent that comparing the intended decomposition of $\text{poly}$ to every other polynomial on the same path up to the root is an extremely time-consuming, thus not a preferable approach. Instead, we choose only those random $\xi$ values which fulfill the following condition. Let us assume that the coefficient $c_i = \alpha_i \cdot 2^{\beta_i}$ of the current $\text{poly}$ is to be decomposed. The intended decomposition is

\[
\xi + (c_i - \xi) ,
\]

where the integer $\xi$ is selected randomly from the interval $[0, 2^{m}]$. If $\text{poly}$ has several component terms, then the decomposition (2) is valid iff

\[
a^* \leq |\alpha_i| \quad \text{and} \quad |a^*| \leq |\alpha_i|. 
\]

If $\text{poly}$ has a single component term ($i = 0$) and $|\alpha_0| > 1$, then in addition to condition (2), the following must also hold:

\[
\xi \neq 0 \quad \text{and} \quad \xi \neq c_0 .
\]

It is easy to see, that the second condition is necessary, otherwise the proposed decomposition might look like $c_0 = 0 + c_0$ or $c_0 = c_0 + 0$. However, we point out that in case of more complex polynomials, such special decompositions are not only allowed but also generally preferred. This way the number of components and consequently the complexity of the polynomials is rapidly decreased.

2.4.2 Delay

A delay indicates actually a shift in the time domain. It changes the exponents $\beta_i$ of the component term $c_i x^{\delta_i} z^{-\beta_i}$,
of \(poly\). Hence, if \(poly\) is gained by a delay, then each of the component terms must have been delayed by the same \(k\) of clock rates, where \(1 \leq k \leq \beta_{\text{max}}\) and \(k \in \mathbb{N}\). The \(\beta_{\text{max}}\) denotes the maximal number of delays could have possible done to achieve \(poly\). Formally, it is the smallest exponent of \(z\) of the component terms:

\[
\beta_{\text{max}} = \min_{1 \leq i \leq n} \{ \beta_i \} .
\]

Consequently, we can decompose \(poly\) as a delayed sub-polynomial only if \(\beta_{\text{max}} < 0\). Occasionally, however, this theoretical upper bound might be to great to be realized efficiently in a hardware. Thus, \(\beta_{\text{max}}\) must not exceed \(\beta_{gw}\)—the maximal number of delays that can be efficiently realized—that is \(\beta_{\text{max}} \leq \beta_{gw}\). Then a random factor \(k\) is taken from the interval \([1, \beta_{\text{max}}]\), and the attribute \(poly_1\) of the child nonterminal is defined as follows:

\[
poly_1 = c_1 x^{\delta_1} z^{-\beta^*_1} \cdots + c_n x^{\delta_n} z^{-\beta^*_n} ,
\]

where \(\beta^*_i = \beta_i - k\) for each \(1 \leq i \leq n\). The delay factor \(k\) is then assigned to the attribute value of the terminal \(\text{int}\).

### 2.4.3 Shift

The current polynomial \(poly\) can be the result of a shift operator, iff it can be decomposed as a product of a sub-polynomial \(poly_1\) and a constant factor \(2^k\) for some \(k \neq 0\), \(k \in \mathbb{N}\). That is holds for each coefficient

\[
\alpha_i 2^\gamma = 2^k \cdot \alpha_i 2^{\gamma + k} .
\]

Consequently, \(poly\) we first look for the maximal number of shifts, \(\gamma_{\text{max}}\), that we can ‘undo’ while decomposing. It is actually the greatest number \(2^{\gamma_{\text{max}}}\) possible to factor out of each component term \(\alpha_i 2^\gamma x^{\beta_i} z^{-\beta_i}\). Formally,

\[
\gamma_{\text{max}} = \min_{1 \leq i \leq n} \{ \gamma_i \} .
\]

Based on this, \(poly_1\) is determined by:

\[
poly_1 = \alpha_1 2^\gamma x^{\delta_1} z^{-\beta_1} \cdots + \alpha_n 2^\gamma x^{\delta_n} z^{-\beta_n} ,
\]

where \(\gamma_i = \gamma_i - k\), for \(1 \leq i \leq n\), and \(k\) is an integer randomly generated in the interval \([1, \gamma_{\text{max}}]\). The actually selected shift factor \(k\) is then assigned to the attribute value of the terminal symbol \(\text{int}\). Nonetheless, we generally choose \(\gamma_{\text{max}}\) for shift.

### 2.4.4 Multiplication

If we want to apply the rewriting rule \(\text{EXPR} \to \text{mul} \text{EXPR} \text{EXPR}\), then our task is to decompose \(poly\) as a product of two sub-polynomials \(poly_1\) and \(poly_2\). In order to determine \(poly_1\) and \(poly_2\), we first have to resolve \(poly\) as a product of the irreducible polynomial:

\[
poly(x) = c \cdot p_1 \cdots p_s ,
\]

where \(c = c_1 \cdots c_r\) is some constant integer. This process is called factorization. For instance in case of a polynomial \(2x^3 z^{-1} + 8x^2 z^{-1}\) it yields the following irreducible sub-polynomials and constant factors:

\[
\begin{align*}
2x^3 z^{-1} + 8x^2 z^{-1} & \quad \left\{ \begin{array}{l}
c = 2 \\
p_1 = xz^{-1} \\
p_2 = xz^{-1} \\
p_3 = xz^{-1} + 4z^{-1}
\end{array} \right.
\end{align*}
\]

generally the factorization is an extremely complex problem, and in our case it is especially difficult, since the polynomials of FIR-structures are defined over two variables, namely \(x\) and \(z\).

Let us recall at this point that in digital signal processing \(z\) refers to them convolution in the discrete time domain. Therefore, according to principle of causality (cf. Section 2.5), polynomials may only be multiplied if the exponents of \(z\) are the same in every component terms. Thus, the decomposition of \(poly\) into a product is allowed only if the condition \(\beta_i = \beta_0\) holds for each component term \(c_i x^{\delta_i} z^{-\beta_0}\), \(0 \leq i \leq n\). In other words this means that we can factorize \(poly\) as it had only one variable \(x\). In order to solve this reduced problem we applied in the computational algebra well-known Berlekamp-Zassenhaus algorithm [1–3].

Hence, if we can determine the irreducible sub-polynomials of \(poly\) as in Equation (2), then we have to distribute them randomly into two groups, \(poly_1\) and \(poly_2\), respectively:

\[
poly_1 = (c_1)^{k_1} \cdots (c_r)^{k_r} \cdot (p_1)^{k_{r+1}} \cdots (p_s)^{k_{s+1}} ,
\]

and

\[
poly_2 = (c_1)^{k_1} \cdots (c_r)^{k_r} \cdot (p_1)^{k_{r+1}} \cdots (p_s)^{k_{s+1}} ,
\]

where the \(k_i^r \in \{0, 1\}\) is a equally distributed random integer and \(k_i^r = 1 - k_i^s\) for each \(1 \leq i \leq r + s\). That is, we take one constant factor or irreducible polynomial after the other and assign them either to the first or the second group. The target group is selected by “tossing up a coin”.

Finally, we obtain \(poly_1\) and \(poly_2\) in the required form—as a sum of component terms—by factoring up the irreducible polynomials of the corresponding groups.

### 2.5 Hardware-specific parameters and constraints

Besides the current value of \(poly\), the parameters and constraints listed in also Table 1 play an important role in the tree generation. The constraints refer to requirements
originating from the hardware synthesis: like the word-
width of the system’s input channel, or the maximal word-
width of the multipliers. Any violation of these constraints
would lead to an error during synthesis. The optimization
parameters, however, impact on the tree generation and
rule selection. More precisely, setting the parameter Delay-
Priority to prior leads to preference of delay over the
other operators. The value normal means no preferred
treatment. This means that if it is possible to apply a delay
during the tree generation, then it is always selected regard-
less what other rule could have also been applied. The same
holds for Shift-Priority. The parameter Delay-Method de-
defines how the amount of the delays is determined. The greedy approach implies that always the greatest pos-
sible delay is chosen, while in case of a stepwise method
at most Max-Delay-Interval delay is performed at a time. In
the core of evolFIR we have encountered further constraints
as well:

- **Causality:** according to (1), it means that no future sig-
nal can be used in the computation of the filter’s output.
To respect this, we had to ensure that the only opera-
tion that is allowed to decrease the order—the greatest
exponent of $z$—of a sub-polynomial is delay. Strictly
speaking, it implies, that a mul can only be applied
if the exponents of $z$ are the same in each component
term of the particular poly.

- **No feedback is allowed:** relating this to the block di-
agrams means that they must be acyclic. Adapting
this to the derivation trees of evolFIR it implies that
we must ensure that every node on the path from the
root of a tree to a leaf has a different poly value.

- **Reduced redundancy I:** the number of component
terms is always minimal. This means, that expression
like $4x + 2x$ never occur, since the component terms
of the same degree and order are always summed up,
i.e. $6x$ stands for $4x + 2x$.

- **Reduced redundancy II:** tree nodes with the same poly
value are mapped to the same node in the block dia-
ogram.

<table>
<thead>
<tr>
<th>HW Constraints</th>
<th>HW Optimization Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-word-width</td>
<td>Shift-Priority</td>
</tr>
<tr>
<td>Max-Delay-Interval</td>
<td>Delay-Priority</td>
</tr>
<tr>
<td>InWordWidth_MULTI_Single</td>
<td>Delay-Method</td>
</tr>
<tr>
<td>InWordWidth_MULTI_Double</td>
<td>Allow-Multiplication</td>
</tr>
</tbody>
</table>

**Table 1. Hardware-related configurable options of evolFIR**

- **Unambiguity of nodes:** the attribute poly is employed
to identify the nodes of our derivation trees. Hence, it
must be prohibited that different notations are used for
the mathematically same polynomial. That is, it is not
permitted to assign first the value $4x^2 + xz^{-1}$ to an in-
stance of poly and then $xz^{-1} + 4x^2$. Unambiguity is
achieved by reducing the component terms of a partic-
ular polynomial in the sense of the previous point, and
sorting them in some predefined order.

### 2.6 Reducing redundancy

Our motivation was efficiency reasons, since a derivation
tree representing a certain FIR structure may contain several
thousands nodes, i.e. may be rather memory-consuming.
We also had to eliminate redundant parts from the result-
ing FIR structure in order to fulfill the requirements of re-
duced redundancy. Principally, both cases require reduction
of the tree individuals. Above of all, we have to realize
these constraints such that the consistency of the derivation
tree is maintained during the whole evolutionary process.
We achieve this in two ways: by using the abstract repre-
sentation of the derivation trees and by using links between
the subtrees describing the same partial polynomial (cf. Fig-
ure 4). The abstract representation of derivation trees basi-
cally means that we cut off (or more precisely, do not insert)
the leaves labeled with terminal symbols. These leaves indi-
cate the derived words but also identify the rewriting rules
applied at the particular points of the tree. This information
has generally no practical importance in the course of the
evolutionary process, yet it is elementary in obtaining
the block diagram of the filter from the best derivation tree.
Therefore, in order to be able to reconstruct the omitted
nodes at any time, we store an identifier of the correspond-
ing rewriting rule (like ADDER or DELAY in a synthesized

![Figure 4. abstract-linked derivation tree](image-url)
attribute). Recalling the rewriting rules of the grammar, we can see that the memory consumption can be reduced to approx. its 2/3 in this way.

In case of the linked representation we go even further and eliminate not only leaves, but whole subtrees in order to avoid undesired redundancy. We want to prevent that a derivation tree contains multiple decompositions of the same sub-polynomial. Therefore, the construction of a new subtree for a specific \( poly \) is always preceded by a search for an existing subtree representing the same sub-polynomial. In case of success, only a link pointing to the existing subtree is inserted instead of creating a new—probably different, thus redundant—subtree.

In case of abstract-linked representation we have no trees anymore but special directed graphs (see Figure 4). Even though these graphs are surely acyclic, in the evaluation of synthesized attributes. Namely, according to the \( L \)-attributed properties of our base attribute grammar, the synthesized attributes of a child node may only influence the synthesized attributes of its parent node. From this follows that in case of an alteration of a subtree, the only attributes needed to be recomputed are the ones on the path upwards to the main root of the tree.

Unfortunately, due to links between nodes in the individuals a child node may actually have several parents. Thus, in case of manipulation of a subtree (subgraph) not only one path, but every possible path up to the main root must be traversed in order to reevaluate the affected synthesized attributes. Obviously, reducing the memory-consumption at the price of significantly increased computational costs is not advantageous. We manage this problem by skillfully reducing the critical attribute dependencies. The clue is the following: in evaluating attributes we only take the attributes of “real” subtrees into account and ignore the ones of “linked” subtrees. Figuratively speaking, it is as if we had reduced the graph to its spanning tree. The attribute dependencies in a spanning tree can be defined in a same manner as in a conventional derivation tree. Thus, keeping them correct requires in the most cases not considerably more effort as normally. A positive side effect of dealing with the attributes in this way is that they accurately reflect the properties of the described redundancy-free block diagrams. It places us in a much better position during the optimization process. All tree-manipulating operators of the evolutionary process rely on this tree generator and so the search space is reduced exactly to the set of valid block diagrams.

3 Evolutionary process

The evolutionary engine of \textit{evolFIR} is realized upon the evolutionary function library OOEA [8]. We have adapted the tree operators of OOEa to derivation trees while taking the previously discussed restrictions into consideration.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{evolutionary_cycle.png}
\caption{Evolutionary cycle}
\end{figure}

These operators have also been enhanced to be able to interact with the random tree generator of the grammar and to deal with the attributes.

During the optimization process, we iterate an the evolutionary cycle depicted in Figure 5 several times. From the initial population, we pick out some individuals according to the selection method specified by the evolutionary parameter \textit{parent selection}. These individuals are the parents, which can be modified in order to gain new individuals. The individuals in the \textit{offspring} build the basis of the \textit{survivor selection}. Here, we find both the new individuals created by the evolutionary operators and the unchanged individuals of the original population. Which individuals actually survives and makes it to the next generation varies due to the mechanism of survivor selection. In order to ensure that already found good individual are not got lost, we can secure them with the help of the \textit{elite selection}.

3.1 Mutation

In the case of mutation, changing a subtree practically means the generation of a new subtree with the \textit{same} root. That is, the label and the value of corresponding inherited attributes must be the same, especially holds for the attribute \( poly \). This way it is ensured that the newly constructed subtree is an alternative description of the same partial transfer function. (The synthesized attributes are of no interest in the node selection, since their value will surely change after the tree generation.) Having the same root, the substitution of subtrees is then equivalent with performing another alternative derivation from the given nonterminal symbol. After the insertion of the new subtree, the nodes along the path from the mutation point to the root have to be updated. This means that several synthesized attributes in these nodes, like depth or size of tree, must be recalculated, because their value is influenced by the values of the synthesized attributes of the new subtree.
3.2 Crossover

In case of crossover, first a random node—i.e. a subtree—is selected. Then we perform a random search in the other parent tree for a same node in the previous sense. Then the subtrees rooted in these nodes—i.e. alternative decompositions of the same transfer functions—are swapped. The nodes in the paths to the roots must be updated in both tree.

3.3 Fitness function

The quality of individuals is measured by the following fitness function:

\[ \varphi = \frac{1}{\text{depth}} \times \frac{1}{\text{size}} \times 10^5, \]

where depth and size denotes the depth and size of the tree-individual, and the factor $10^5$ is used to prevent computational errors due to round-offs. With the help of this fitness function we can favor trees that are not too deep, and do not contain too many nodes (i.e. block elements).

A great advantage of using attributed derivation trees is that they can be evaluated in constant time. Since the depth and size of the trees are realized as synthesized attributes, their values are always computed at the moment of fitness evaluation. Thus the fitness value can always be determined in constant time.

4 Results

To test our system we took a randomly generated transfer function containing 101 linear component terms, that is it has a maximum delay of 100. We used the following parameter settings of the evolutionary engine: the population contained 100 individuals, from which we selected 70 as parents with rank-based selection. By means of the operators 70 new offsprings were created, such that we set the mutation rate to 50%. The survivor-offspring are selected by best selection. The search space of this problem is extremely large, therefore we have performed 2000 generations in each test run on a machine with AMD Opteron™ Processor 246 with CPU 1993 MHz. The runs has been made individually in order to ensure full CPU capacity.

We examined in the first row the impact of the hardware-specific parameters enlisted in Table 1. We declared two groups of experiments: in one of them we allowed the usage of multipliers (i.e. Allow-Multiplication=yes), while in the other one the evolFIR worked without them (Allow-Multiplication=no). The specific runs in both group were performed with four possible combinations of the parameters Shift-Priority and Delay-Priority. Moreover, we limited the maximal amount of delays to 16 and used the stepwise variant of the Delay-Method. The amount of the used block elements shift, delay, add and mul in the best individuals are depicted in Figure 6.

In order to get a feeling about quality of our results, we synthesized the best individuals on Virtex™-II Pro FPGA platform of Xilinx\(^1\). Our results created with multipliers are compared to the filter structure gain by using MAC (Multiply Accumulate) architecture. These are depicted in Table 2. The results without multipliers are set in relation with the filter structure constructed with Distributed Arithmetic architecture in Table 3.

5 Conclusion

In this paper a novel approach for optimizing FIR structures using DTGP was introduced. The proposed method, evolFIR, optimizes FIR structures already in the phase of the logic design optimizing the number and connection of

\(^1\)http://www.xilinx.com/products/silicon_solutions/fpgas/virtex
the function blocks. The possible valid connections of the function blocks are defined by an attribute grammar. Therefore additional function blocks, or different types of connections can be added to the system without the necessity of changing the program. Only the grammar has to be replaced, or extended.

The essential issues to deal with during the evolutionary process are as follows. We have to:

- ensure that the individuals always describe exactly the transfer function we look for, that is they are valid. In other words, we have to prevent that the evolutionary operators turn a valid individual into an invalid one;
- assure that the topology fulfills certain hardware-related requirements known from the subsequent levels of circuit and physical design, such as the limited number of input channels or restricted word-width of the dedicated block element;
- recognize the subgraphs describing the same function and reuse them in order to reduce redundancy.

Since the evolutionary process is guided by an attribute grammar, no invalid individuals are created, and the fitness values are computed as attributes. This means the computational costs are reduced radically in compared to a simple tree representation. Therefore, no time-consuming traversal of the tree is required. Moreover, when the algorithm is running, only a small portion of these factors need to be recalculated. Namely, those which are located on the path leading from the root to the the newly generated subtree. Other plans are to experiment with different parallelization techniques.

References