A Java Simulation Tool for Fixed-Point System Design

S. Wijaya, A. Cantoni
Western Australian Telecommunications Research Institute
School of Electrical, Electronic, and Computer Engineering, The University of Western Australia, Perth, WA Australia
wijayas@watri.org.au, cantoni@watri.org.au

ABSTRACT
The realisation of signal processing algorithms in fixed-point offers performance advantages over floating-point realisations. However, the task is widely acknowledged to be tedious, error prone, and time consuming. In this paper, we propose a systematic approach to automate fixed-point system design. The technique generates fixed-point parameters that satisfy a precision constraint imposed on the primary output of the algorithm to be realised. The development of a simulation framework based on this analysis allows fixed-point designs to be generated in a shorter time frame. The effectiveness of the approach and framework is demonstrated through the implementation of an Erbium-Doped Fibre Amplifier (EDFA) control algorithm in fixed-point.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids

General Terms
Design

Keywords
Fixed-point algorithm, digital design, automation tool, Data Flow Graph (DFG)

1. INTRODUCTION
The task of manually converting signal processing algorithms to fixed-point system is challenging with limited tool support [1-6]. However, despite the effort involved, fixed-point implementation offers substantial advantages such as an improvement in execution speed, lesser memory usage, and reduced hardware cost. For this reason, research has emerged aimed at reducing the time and effort spent on this process through design automation.

Some research has been targeted at specific applications [7-10], while others have been targeted at particular hardware architectures [11]. Commercial software, such as MATLAB, provides a toolbox that can help designers with the development of fixed-point design [12]. While commercial tools assist in the translation of floating to fixed-point design, more often than not they still rely on the developers to manually provide the design parameters. Frequently, they do not easily support optimal word sizes for individual variables.

SimuTools'09, *March 2–6, 2009, Rome, Italy*.

Much research has been targeted at systems that are developed in C [3, 4], as C is considered by many as the universal coding language. This has led to the development of language extensions that characterise fixed-point numbers and operations in C [4]. Traditionally, fixed-point implementations are directed for DSP applications, where a uniform word length is chosen to satisfy system requirements. However, it has been realised that unlike DSP, implementation on dedicated hardware is not restricted to a pre-defined word length. Thus, the focus of research in this area has turned to accommodating multiple bitwidth word selection [12, 13]. Various approaches have been used in floating to fixed-point automation schemes including simulation-based exhaustive search [8, 14]. However, these approaches are not practical as they are computationally intensive to perform.

In this paper, the proposed approach for fixed-point design automation is targeted for hardwired FPGA implementation of systems that can be represented with a data flow graph (DFG). Hardwired FPGA implementation offers designers with the freedom to optimise the system to achieve the design goals of specific applications. The approach proposed in this thesis for floating to fixed point design automation consists of several components:

- Data Flow Graph (DFG) representation of algorithms
- Range analysis
- Precision analysis
- Automated word size selection for variables

The analysis of algorithms via their representation as a graph is a common approach which has been used in [5, 6]. Range estimation involves determining the dynamic range of the variables in the system. Two popular techniques found in the literature are the worst case estimation [15, 16] and the statistical approach [17]. This work presented in this paper utilises the worst-case estimation technique since simulation results are often not reliable for accurately predicting the behaviour of the actual system. Precision analysis is concerned with the selection of word lengths for variables in the system. Various techniques found in the literature for precision analysis include automatic differentiation [18] and non-linear optimisation [5]. The issue of error propagation is considered in [5], however the approach used to determine the word length of variables is different to that adopted in this paper.

A Java realisation is developed according to the proposed approach for fixed-point design automation. The application of this simulation utility is demonstrated with a practical design task, i.e. the realisation of Erbium-Doped Fibre Amplifier (EDFA) control algorithm in fixed-point.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

The paper is organised as follows. In Section 2, the formal notation and mathematical formulation for fixed-point representation of numbers are introduced. In Section 3, the proposed approach to convert of an algorithm, given in floating-point or infinite precision, to fixed-point through automation is presented. In Section 4, an example of practical implementation of the Java tool is presented. In Section 5, concluding remarks are provided.

2. Fixed-Point Representation and Analysis

2.1 Fixed-Point Notation

A fixed-point number contains key parameters that are crucial for its representation. In this paper, the fixed-point representation of a real number X, denoted as $\overline{X}$, consists of several components:

- The sign bit S_x

  In set notation, the sign is given by $S_X \in \{'0','1'\}$. A sign bit value of '0' is used to indicate a positive number, while '1' is used to indicate a negative number.

- The fraction bits $X_1 \ \ldots \ X_{N_x}$

  Each fraction bit X_i is given by $X_i \in \{'0','1'\}$ where $i \in \mathbb{N}^+$ and $1 \le i \le N_x$. The fractional part is also frequently known as the mantissa, and its function is to store the precision bits of the number. This representation assumes an implicit decimal point to the left of the mantissa's most significant bit. Each bit carries a weighted value of 2^{-i}, where i signifies the position of the bit and $i = 1$ denotes the most significant bit. The decimal expansion of the fractional value is calculated as a sum of products $X_1.2^{-1} + X_2.2^{-2} + X_3.2^{-3} + \ldots + X_{N_x}.2^{-N_x}$. The magnitude of the fractional part is always less than 1.

- The scaling factor E_x

  The scale factor E_x is an integer $E_x \in \{\ldots, -1, 0, 1, 2, \ldots\}$ that is associated with a variable and is used to scale the fractional values of the fixed-point representation. The decimal value of the fixed-point representation can be obtained by multiplying the expansion of the mantissa with a factor of 2^{E_x}. This field is implied and is not physically included in the representation. System developers are expected to internally keep track of the change in scaling factors when designing the system.

- The word length N_x

  The word length N_x, where $N_x \in \{0, 1, 2, \ldots\}$, signifies the number of bits used for representation. The value of N_X does not include any sign information.

S_X	X_1	X_2	…	…	…	…	X_{N_x}	x 2^{E_X}

Figure 1. Fixed-point representation of X

Diagrammatically, the fixed-point approximation of real number X as a signed magnitude is illustrated in Figure 1. A decimal point is assumed between S_x and X_1.

2.2 Fixed-Point Mathematical Formulation

Mathematically, the signed magnitude representation of a real number X is given by:

$$\overline{X} = M_x 2^{E_x} = S_X \left(\sum_{k=1}^{N_x} X_k 2^{-k} \right) 2^{E_x} \quad (1)$$

In (1), $\overline{X}$ represents the fixed-point approximation of the real number X, S_x signifies the value of the sign bit, and $\left(\sum_{k=1}^{N_x} X_k 2^{-k} \right)$ is the decimal value of the fractions bits. This notation assumes that the value of the sign bit S_x is 1 for positive numbers and -1 for negative numbers. The value of M_x can be obtained by multiplying the sign bit and the decimal value of the fraction bits. The word length is limited by truncating the least significant bits of the mantissa.

The signed magnitude representation has a bounded error as shown below:

$$\Delta X = X - \overline{X} = S_X \left(\sum_{k=N_x+1}^{\infty} X_k 2^{-k} \right) 2^{E_x}$$

The maximum actual error is given by (2)

$$\left| X - \overline{X} \right| < 2^{-N_x} 2^{E_x} \quad (2)$$

since $\sum_{k=N_x+1}^{\infty} X_k 2^{-k} < 2^{-N_x}$.

2.3 Error Bound Analysis

The principle operations of the Java tool is based on an investigation of error bounds for primitive arithmetic operations, including addition, subtraction, and multiplication. The aim of this analysis is to determine the minimum word length for operands given the precision required at the output, denoted as r_c.

The error bound analysis is derived from the following statements for approximating X as a fixed-point variable $\overline{X}$:

- The maximum value of $\overline{X}$, denoted by $\overline{X}_{\max}$, is 2^{E_x}.
- From (2), the maximum actual error in the representation of X, denoted by $\Delta X_{\max}$, is $2^{-N_x} 2^{E_x}$.

2.3.1 Fixed-point multiplication

In this section, error bound analysis for fixed-point multiplication operation is presented. Consider real numbers X, Y, and Z, where Z is the product of X and Y.

$$Z = XY = (\overline{X} + \Delta X)(\overline{Y} + \Delta Y)$$

The error in representing Z in fixed point can be derived as

$$\left| Z - \overline{Z} \right| = \left| XY - \overline{XY} \right| = \left| \overline{X} \Delta Y + \overline{Y} \Delta X + \Delta X \Delta Y \right|$$

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

The maximum absolute error in fixed-point multiplication can be derived as

$$\left|Z-\overline{Z}\right|_{\max} = \left|\overline{X}\Delta Y\right|_{\max} + \left|\overline{Y}\Delta X\right|_{\max} + \left|\Delta X\Delta Y\right|_{\max}$$
$$= \overline{X}_{\max}\Delta Y_{\max} + \overline{Y}_{\max}\Delta X_{\max} + \Delta X_{\max}\Delta Y_{\max}$$

Since $2^{-N_X}2^{-N_Y} << 2^{-N_X} + 2^{-N_Y}$, this equation can be simplified as

$$\Delta Z_{\max} = \left|Z-\overline{Z}\right|_{\max} = 2^{E_X}\Delta Y_{\max} + 2^{E_Y}\Delta X_{\max}$$

Given a required output precision of 2^k, where $k = -N_z + E_z$, it is possible to determine the word length of multiplicands that will satisfy this precision.

To obtain a maximum error at the output of $\Delta Z_{\max} = 2^k$ and assuming equal error contributions, it can be shown that

$$\Delta Z_{\max} / 2 = 2^{E_Y}\Delta X_{\max}$$
$$N_X = E_X + E_Y - (-N_Z + E_Z - 1) \quad (3)$$

Using the same derivation, it can be concluded that $N_X = N_Y$.

In summary, performing fixed-point multiplication can be formulated based on these findings. The fixed-point parameters of a multiplication operation $\overline{Z} = \overline{X} * \overline{Y}$ can thus be determined in steps as outlined below.

1. The scaling factor of multiplicands can be determined according to the following rules.

   $$E_X = floor\left\langle \log_2 X \right\rangle + 1,\ E_Y = floor\left\langle \log_2 Y \right\rangle + 1 \quad (4)$$

   Likewise, if the operand is a variable with a dynamic range X ~[min, max], then E_x can be determined with the following rule

   $$E_X = floor\left\langle \log_2\left(\max|X|\right)\right\rangle + 1, \quad (5)$$

   where $\max|X| = \max(|\min(X)|, |\max(X)|)$

2. The scaling factor of $\overline{Z}$ can be determined by the following rule

   $$E_z = floor\left\langle \log_2\left(\max|Z|\right)\right\rangle + 1, \quad (6)$$

   where $\max|Z| = \max(|\min(Z)|, |\max(Z)|)$

3. The dynamic range of the product Z can be determined according to the formulas given below

   $$\min(Z) = \min\begin{Bmatrix}\min(X)*\min(Y), \max(X)*\min(Y),\\ \min(X)*\max(Y), \max(X)*\max(Y)\end{Bmatrix} \quad (8)$$

   $$\max(Z) = \max\begin{Bmatrix}\min(X)*\min(Y), \max(X)*\min(Y),\\ \min(X)*\max(Y), \max(X)*\max(Y)\end{Bmatrix} \quad (9)$$

   Assuming one of the operands (X) is a constant and the other (Y) is a variable, the range of Z can be determined as follows

   $$\min(Z) = X * \min(Y) \quad (10)$$
   $$\max(Z) = X * \max(Y) \quad (11)$$

4. The word length of $\overline{Z}$ can be determined according to the precision criteria for the output. Given a required accuracy of 2^k, where $k = -N_z + E_z$, N_z can be determined by the following rule

   $$N_z = E_z - k \quad (12)$$

5. From (3), the word length of operands $\overline{X}$ and $\overline{Y}$ can be determined according to the following rule

   $$N_X = N_Y = E_X + E_Y - (-N_Z + E_Z - 1) \quad (13)$$

   Each register requires an extra bit, added to the word length of each operand to allow for the sign bit.

For fixed-point multiplication, no shift is required prior to operation as there is no risk of overflow and underflow. However, a shift might be required on the result in cases where $E_z < E_X + E_Y$ has resulted from following the steps above. If $E_z = E_X + E_Y - 1$, a logical left shift is to be performed on the mantissa of the fixed-point number $\overline{Z}$. The number of shifts is $E_X + E_Y - E_z$ bits.

2.3.2 Fixed-point addition/subtraction

Fixed-point parameters for addition/subtraction operations can be determined with similar analysis and rules as outlined in this section. Consider real numbers X, Y, and Z, where Z is the sum of X and Y. It can be shown that the error in representing the sum in fixed-point (ΔZ) is the sum of the quantization error of the operands. The error in the addition operation can be derived as follows:

$$Z = X + Y$$
$$= \overline{X} + \Delta X + \overline{Y} + \Delta Y = \overline{Z} + \Delta Z$$
$$\Delta Z = \Delta X + \Delta Y$$

The maximum error in representing Z, $\Delta Z_{\max}$, can be derived as follows

$$\Delta Z_{\max} = \left|Z-\overline{Z}\right|_{\max} = \Delta X_{\max} + \Delta Y_{\max}$$
$$\Delta Z < 2^{-N_X+E_X} + 2^{-N_Y+E_Y}$$

Assuming equal error distribution $2^{-N_X+E_X} = 2^{-N_Y+E_Y}$, it can be shown that (0.0.7)

$$\Delta Z < 2\left(2^{-N_X+E_X}\right) = 2\Delta X$$

From this analysis, it can be concluded that, in order to obtain a precision of 2^k, where $k = -N_x + E_x$, the operands must be represented with a minimum precision of 2^{k-1}.

In summary, the selection of the word length of variables for fixed-point addition operations can be formulated based on this analysis. The fixed-point parameters of each operand in an addition operation $\overline{Z} = \overline{X} + \overline{Y}$ can be determined according to the following steps:

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

1. The value of scaling factors E_X and E_Y can be determined according to (4) and (5).
2. The scaling factor of $\overline{Z}$ can be determined according to the same rule given in (6).
3. The dynamic range of the sum Z can be determined according the rules described in (8)-(11) and by replacing the multiplication operator (*) with an addition operator (+).
4. The word length of $\overline{Z}$ can be determined based on the precision criteria for the output. Given a required accuracy of 2^k, where $k = -N_Z + E_Z$, N_Z can be determined by the following rule

$$N_Z = E_Z - k + 1 \quad (14)$$

Note that $\overline{Z}$ requires an extra bit to accommodate the precision carried by the operands that have a precision of 2^{k-1}.

5. The word length of operands $\overline{X}$ and $\overline{Y}$ can be determined according to the following rules

$$N_X = N_Z - (E_Z - E_X) \quad (15)$$

$$N_Y = N_Z - (E_Z - E_Y) \quad (16)$$

Every register that stores a fixed-point representation contains an extra bit to allow for the sign bit.

6. The number of shifts required by both operands prior to addition is given by $E_Z - E_X$ and $E_Z - E_Y$. These values correspond to the number of logical right shifts to be performed on the operands A and B respectively.

The analysis presented in this section can be applied to both addition and subtraction operations.

2.4 Resolution of Arithmetic Operators

In fixed-point implementation, the number of bits required for each arithmetic operation in the algorithm varies depending on the data widths of its inputs as well as the resolution constraint at the output. Arithmetic modules typically assume a uniform width for its operands. For example, a 12-bit adder assumes its input and output variables to have 12 bits of data width. For this reason, arithmetic shifts are required to align the operands for computation. For a fixed-point operation $\overline{C} = \overline{A} \rho \overline{B}$, where ρ is an operator unit, the resolution of the operator is determined according to the rule

$$bits(\rho) = \max(N_A, N_B, N_C) \quad (17)$$

where $bits(\rho)$ signifies the weight of the operator. This provides sufficient width to execute the operation without losing the required precision.

This attribute is determined only after $N_A, N_B,$ and N_C are resolved. In order to achieve a uniform width for computation, input operands whose word lengths are less than $bits(\rho)$ are adjusted by padding their mantissa with zeros.

3. THE JAVA SIMULATION TOOL

In this section, a technique that can be used to automatically determine the fixed-point parameters of a given algorithm is proposed. This method relies on the DFG representation of algorithms and a graph traversal algorithm to achieve design automation. The proposed technique, in conjunction with the error bound analysis presented in section 2.3, allows the development of the Java simulation tool capable of generating a design satisfying the precision constraint r_c. The input to the tool is described in a text file, according to pre-defined syntax rules. The current implementation is text-based, however a graphical interface can be easily adopted using on the same principle of operations.

A pipelining feature integrated in the tool will further assist developers to meet the speed constraint of the system. This feature, however, will not be discussed in this paper.

3.1 Data Flow Graph Representation

In this paper, a data flow graph(DFG) representation is utilised to automate the translation of algorithms into fixed-point. This diagram aids in the visualisation of algorithms by providing a complete layout of operators, operands, as well as the interconnection between them. A DFG includes the direction of the data flow and should not contain cycles.

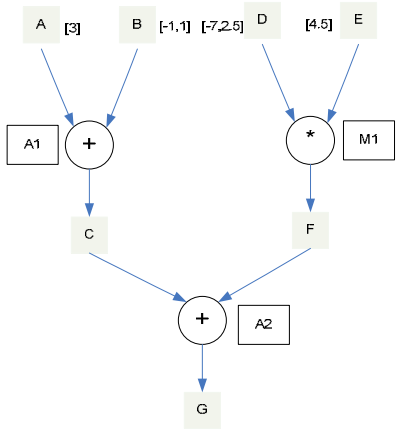

Figure 2. A sample DFG representation

For clarity in notation and presentation of the graph traversing algorithm, the operator and operand nodes are distinguished by the following notation:

- $\gamma \in Operand$
- $\rho \in Operator$

In Figure 2, $Operand = \{A, B, C, D, E, F, G\}$ and $Operator = \{A1, M1, A2\}$.

For $\gamma \in Operand$, γ must be a member of one of the following subsets:

- $\gamma \in InputVar$ – input variables
- $\gamma \in Constant$ – inputs that are constants
- $\gamma \in IntermediateVar$ – intermediate variables

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

In Figure 2, $InputVar = \{B, D\}$, $Constant = \{A, E\}$, $IntermediateVar = \{C, F, G\}$

An operand node γ is characterised by the following attributes:

- The dynamic range of γ, $range(\gamma)$
  This attribute represents the dynamic range of the operand γ in infinite precision. For $\gamma \in InputVar$ or $\gamma \in IntermediateVar$, this attribute is characterised by $[\min, \max]$ that represents the lower and upper bound value of γ. For $\gamma \in Constant$, $range(\gamma)$ is characterised by $[value]$ that corresponds to the numerical value of the constant. The dynamic range of $\gamma \in InputVar$ and $\gamma \in Constant$ is specified as DFG inputs. The dynamic range of an intermediate variable can be derived using the conservative range propagation technique described in equations (8) - (11) for fixed-point multiplication. Similar rules can be used for fixed-point addition.
- The scaling factor of γ, E_γ
  The scaling factor directly corresponds to the fixed-point parameter E_γ of γ, which has been defined in Section 2.1. The rules to obtain the scaling factor E_γ of operand γ are described in equation (4) - (6).
- The word length of γ, N_γ
  The word length of the operand γ in a DFG is directly related to the parameter N_γ, that characterises the word length for the fixed-point representation of γ. The value of N_γ does not take into account an extra bit that is allocated for the sign. The word length N_γ of operand γ can be selected using the rules described in (12) - (13) for fixed-point multiplication operations. Similarly, the rules given in (14) - (16) can be used to select the word length of the operand γ for fixed-point addition operations.

For $\rho \in Operator$, ρ must be a member of one of the following subsets:

- $\rho \in AddSub$ – adder/subtractor operator
- $\rho \in Multiplier$ – multiplier operator

Operator nodes are always connected to two input nodes and one output node. In Figure 2, $AddSub = \{A1, A2\}$ and $Multiplier = \{M1\}$.

An operator node ρ is characterised by attribute *bits(* ρ *).* This attribute represents the bit resolution (weight) of the operator. For example, in fixed-point computation, a 16-bit adder might be required for one operation while a 12-bit one might be sufficient for another. This attribute defines the minimum bits required to perform the operation ρ without losing the precision of the result. The formula used to determine this attribute is given in (17).

Each operator node ρ is associated with input and output variables, which are described by the following notation:

- $v_o(\rho)$ - an output node that is connected to ρ. The DFG used in this paper is arranged such that an operator is connected exclusively to one output.
- $\vec{v}_i(\rho)$ - a vector of input nodes related to ρ. An operator ρ is associated with at least one input node.

For example, in Figure 2, $v_o(A1) = \{C\}$ and $\vec{v}_i(A1) = \{A, B\}$.

For clarity in the sorting procedure, the following notation is used to describe the sequence of vertices related to any node $v \in V$:

- *precedingAdj(v)* – a set containing preceding adjacent vertices
- *subsequentAdj(v)*– a set containing subsequent adjacent vertices

For example, in Figure 2, *precedingAdj(A2)* = {*C*,*F*} and *subsequentAdj(F)*={*A2*}.

3.2 DFG Traversal Algorithm

The DFG traversal algorithm proposed in this section allows fixed-point design to be automated. Namely, the critical parameters of each variable in the algorithm implementation satisfying the output resolution constraint r_c are calculated and produced.

The proposed technique involves traversing the DFG which represents the target algorithm such that all $\rho \in Operator$ and $\gamma \in Operand$ are visited. The fixed-point parameters associated with each operator and operand, such as E_γ and N_γ, are then determined during this traversal. The process can be explained in steps as follows:

1. Firstly, the nodes of the DFG are sorted according to their arithmetic evaluation order.
2. Once the nodes are sorted, the DFG is traversed in the forward direction. During traversal, the attributes of the operator output are determined, including $range(v_o(\rho))$ and $E_{v_o(\rho)}$ for all $\rho \in Operator$.
   - $range(v_o(\rho)) = f_1\left(range\left(\vec{v}_i\right)\right)$
   - $E_{v_o(\rho)} = f_2\left(range(v_o(\rho))\right)$

   where f_1 and f_2 are some given functions.
3. The word length of the output is determined based on the output resolution criteria r_c.
4. The DFG is traversed in the reverse direction and the word length of operator inputs $N_{\vec{v}_i(\rho)}$ for all $\rho \in Operator$ are determined such that the output resolution criteria r_c is met.

   $$N_{\vec{v}_i(\rho)} = f_3\left(N_{v_o(\rho)}\right)$$

   The resolution of ρ is determined such that

   $$bits(\rho) = f_4\left(N_{v_o(\rho)}, N_{\vec{v}_i(\rho)}\right)$$

   where f_3 and f_4 are some given functions.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

In the subsections that follow, each of these steps will be discussed in more detail.

3.2.1 Topological Sort

Evaluating arithmetic expressions as DFGs requires the order of precedence to be maintained to avoid having incorrect results. The order of evaluation is such that the topmost leaves of the DFG are examined first and its root examined last. In Figure 2, nodes A and B are to be evaluated before node $A1$, and $A1$ before C.

In developing the Java tool, a sorting algorithm has been utilised to arrange computation nodes according to their evaluation order. Sorting has been the subject of extensive research in computer science literature due to its significance. Various sorting techniques have been developed as a result [19], one of which is termed *topological sort*. Topological sort involves a linear ordering of all nodes in a directed acyclic graph (DAG) with precedence constraints. It defines the sequence for traversing a graph such that a node $v \in V$ is visited only after all preceding nodes *precedingAdj(v)* have been visited.

The topological sort algorithm [19] can be summarised as follows:

1. Find a node v_k that has no successors, i.e. $subsequentAdj(v_k) = \{\}$ or $|subsequentAdj(v_k)| = 0$
2. Delete this node v_k from the graph and all edges connected to this node, i.e. $e_i \in \{(v_j,v_k), (v_m,v_k),\ldots\}$
3. Insert node v_k into a queue/list

The result of applying this sorting algorithm is a list that defines the order in which the graph should be traversed. The use of the topological sort algorithm allows the node entries described in the input text file to be given in any order.

3.2.2 DFG Downward Traversal

Computation nodes are visited according to their topological order, from the leaf nodes to the root. During graph traversal, fixed point parameters are determined based on the rules developed in Section 2.3. The following procedures are executed when the graph is traversed downward:

For all $\gamma \in$ *InputVar* or $\gamma \in$ *Constant,*

1. Identify *range(* γ *)*
2. Determine the scaling factor E_γ based on *range(* γ *)*

For all $\rho \in$ *Operator,*

1. Determine the dynamic range of the operator's output $range\left(v_o(\rho)\right)$ based on the range of the operator's input $range\left(\vec{v}_i(\rho)\right)$.
2. Determine the scaling factor of the operator's output $E_{v_o(\rho)}$ based on their respective dynamic range $range\left(v_o(\rho)\right)$.
3. Detect overflow if $\rho \in$ *Adder*

When the graph is traversed downward, different procedures are carried out depending on whether the node is an operand or an operator. The Java tool assigns the scaling factor for every input variable and constant based on its given range. For every operator encountered during downward traversal, the tool estimates the dynamic range and determines the scaling factor associated with the operator's output.

For example, consider the operation $C=A+B$. Assume operand A has a dynamic range of [10,20] and B [-15,5]. A graph G that represents this operation will consist of 4 nodes, $V = \{A, B, C, Add1 \mid A, B, C \in Operand, Add1 \in Operator\}$. The order of traversal is such that input nodes are visited before output, i.e. $A \rightarrow B \rightarrow Add1 \rightarrow C$. The following operations are performed at each node:

1. Visit A, assign $E_A = 5$
2. Visit B, assign $E_B = 4$
3. Visit *Add1*, assign *range(C)*=[-5, 25], assign $E_C = 5$
4. Visit C, assign $E_C = 5$

3.2.3 Determination of Output Word Length

Given a resolution constraint r_c for the output variable Z, the output word length can be determined using (18).

$$N_Z = floor\left\langle \log_2\left(\frac{\max\left(\left|\max(Z)\right|,\left|\min(Z)\right|\right)}{r_c(Z)}\right)\right\rangle + 1 \qquad (18)$$

The number of bits allocated for Z must allow the maximum possible magnitude to be represented. As a signed magnitude representation is used, the maximum amplitude can be obtained by comparing the positive and negative extremes.

3.2.4 DFG Upward Traversal

Once the word length is of system output is determined, nodes in the DFG are visited starting from the root and in reverse order to the initial direction of traversal. As the DFG is traversed, the word-length of operands and intermediate variables are assigned by taking into account the constraint r_c.

The following procedure is performed as each node is visited:

For all $\rho \in$ *Operator* ,

1. Determine $N_{\vec{v}_i(\rho)}$ for all ρ according to
   - $N_{v_o(\rho)}$ and $E_{v_o(\rho)}$
   - The type of operator
2. Determine *bits(* ρ *)* – bit resolution of the operator. The resolution of the operator is determined by choosing the maximum word length of all operands connected to it.

$$bits(\rho) = \max\left(N_{v_o(\rho)}, N_{\vec{v}_i(\rho)}\right)$$

If operator ρ has a branched input, a uniform word length is selected by examining all possible paths and choosing the maximum word length among the values presented during upward traversal.

3.3 Textual Representation of the Tool

The information on all nodes of the DFG, including their parameters, is incorporated in a text file. Each line in the input file describes a node in the DFG. Each field of information in each entry is separated by a semicolon. Some keywords are used to

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

differentiate various types of nodes. For the DFG depicted in Figure 2, some examples of node entries are given below.

```
operand;constant;A;3;A1
operand;inputVar;B,-1;1,A1
operator;adder;A1;A;B;C
```

The first line describes a node A, which is a constant input with a value of 3 and is connected to operator $A1$. The second line describes a variable input B, with a dynamic range of [-1, 1] and is connected to operator $A1$. The third line describes an adder $A1$, which is connected to two input nodes, A and B, and an output node, C.

Similarly, the output of the tool is given as a text. For example, after simulation run, the fixed-point parameters for operand A, B, and $A1$ are given below (for r_c =0.5).

```
Label : A
Is a constant
SF : 2
WL : 5
Value : 3
Neighbour node(s) : A1
```

```
Label : B
Is an input variable
SF : 1
WL : 4
Minimum : -1
Maximum : 1
Neighbour node(s) : A1
```

```
Label : A1
Is an adder
Bits : 6
Neighbour node(s) : A  B  C
```

4. AN EXAMPLE OF SYSTEM DESIGN

In this section, the Java tool is employed to provide a fixed-point realisation of a control algorithm.

4.1 EDFA Control Algorithm

Erbium Doped Fibre Amplifier, commonly abbreviated as EDFA, is one of the more popular optical amplifiers [20, 21]. The EDFA has contributed significantly to the success of Wavelength Division Multiplexing (WDM) technology [22]. This technology allows optical signals of a range of wavelengths, each represented in an individual channel, to be transmitted simultaneously within one optical fibre without interfering with one another.

The dynamic nature of optical networks allows channels in WDM systems to be added or dropped arbitrarily due to network reconfigurations or component failures [23, 24]. Disruptions to the input signal channels can cause the EDFA gain and the power in the surviving channels to fluctuate as a result. As transient EDFA gain excursions can adversely degrade the performance of the system [23-27], it is necessary to implement dynamic control of the pump power in order to mitigate EDFA gain excursions. As a consequence, the design of EDFA pump control algorithms has been the subject of extensive research over the past decade [23, 24, 28, 29].

The EDFA control algorithm considered in this paper has been proposed in [23, 30-32]. In this approach, the suppression of EDFA gain excursions is achieved by adjusting the signal power of the external laser pump, according to measurements of the EDFA total input and output signal power.

The EDFA control algorithm, illustrated in Figure 3, is a closed-loop system which consists of two components, a feed-forward and a feed-back component. The feed-forward component provides a quick adjustment to the pump power as soon as a fluctuation in total input signal power is detected. The feed-back component is a proportional integrator (PI) controller which provides correction to the feed-forward values. The pump control block combines the feed-forward and feed-back components to produce a pump output in such a way that keeps the gain constant. Two types of pump control, additive and multiplicative, are considered. Both are illustrated in Figure 4.

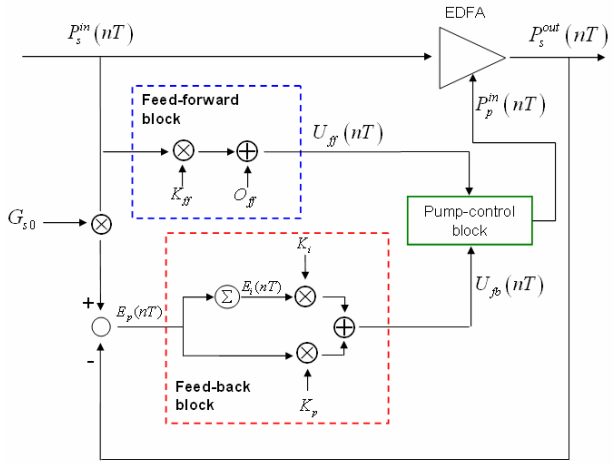

Figure 3.Discrete time EDFA closed loop system

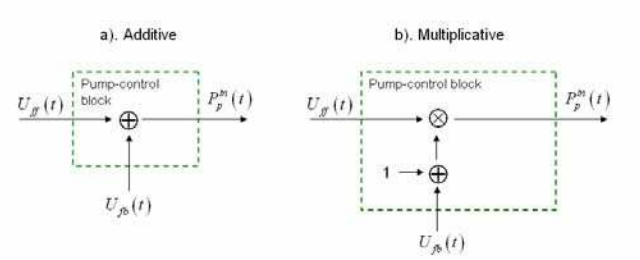

Figure 4. Discrete time pump-control block: a) additive type and b) multiplicative type

The mathematical formulation of the discrete time EDFA control algorithm can be found in [23, 30-32] and is summarised below.

1. Feed-forward component

$$U_{ff}(nT) = K_{ff} P_s^{in}(nT) + O_{ff} \tag{19}$$

where $U_{ff}(nT)$ is the feed-forward component, $P_s^{in}(nT)$ is the input signal power, K_{ff} and O_{ff} are the feed-forward constant and offset respectively.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

2. Feed-back component

$$U_{fb}(nT) = K_p E_p(nT) + K_i E_i(nT) \quad (20)$$

where $U_{fb}(nT)$ is the feedback component, K_p and K_i are proportional and integral constant respectively. $E_p(nT)$ and $E_i(nT)$ represent the proportional and integral error respectively.

3. Error formation

$$E_p(nT) = G_{s0} P_s^{in}(nT) - P_s^{out}(nT) \quad (21)$$

$$E_i(nT) = E_i((n-1)T) + E_p(nT) \quad (22)$$

where G_{s0} is the desired gain. The proportional error is calculated based on the difference between the desired output signal power $G_{s0} P_s^{in}(nT)$ and the actual measured output signal power $P_s^{out}(nT)$. The integral error is the summation of proportional error over time.

4. Pump control block

$$\text{Additive } P_p(nT) = U_{ff}(nT) + U_{fb}(nT) \quad (23)$$

$$\text{Multiplicative } P_p(nT) = U_{ff}(nT)\left(1 + U_{fb}(nT)\right) \quad (24)$$

where $P_p(nT)$ represents the pump power, which is the primary output of EDFA control algorithm.

4.2 Fixed-point Realisation of EDFA Control Algorithm

The realisation of EDFA control algorithm in fixed-point allows an improvement in the execution speed over a floating-point design. The control system is thus able to respond more rapidly to the changes in network configuration.

In this paper, fixed-point realisation of EDFA control algorithm involves three different stages of verification:

- Floating-point simulation
- Fixed-point simulation
- Emulation

In the first stage, a MATLAB Simulink model was developed to simulate the behaviour of the EDFA closed-loop system. The result of the floating-point simulation is used as a reference model for the fixed-point equivalent. In the second stage, the EDFA control algorithm is substituted with the fixed-point model, according to the design generated by the Java tool. The last stage, the EDFA emulation, combines the two components of the EDFA amplification system (the amplifier and the control system), which are implemented on two different platforms. The EDFA control algorithm, which is run on a hardware platform, is coordinated with the software simulation of the EDFA to form a closed-loop system. In this paper, the algorithm is implemented Stratix FPGA in fixed-point using a hardwired approach. Note that the same input signal is used in all the three stages of development.

In this paper, the control parameters are chosen to achieve a desired steady-state gain G_{s0} of 14 dB and a tolerance of ∂G_s =±0.1 dB. Using the formulation of EDFA dynamic model found in [23,31-33], it can be shown that in order to achieve this gain precision, the pump power has to maintain a resolution of 1.2059 mW. This value is specified as the output precision criteria r_c when designing the EDFA control algorithm in fixed-point using the Java tool.

The representation of the feedback component of the EDFA control algorithm as a DFG presents a challenge as the summation operator introduces a cycle in its graph representation. This challenge is addressed by making an assumption that $E_{i,n}$ and $E_{i,n-1}$ are independent variables. The dynamic range of the $E_{i,n-1}$ can be estimated through the EDFA simulation environment, which allows the values of intermediate variables to be read.

The results of floating point simulation are shown in Figure 5 and Figure 6. It is shown that EDFA steady state gain of 14 dB is achieved with the pump power depicted in Figure 6. A glitch observed at t = 5ms and t =10ms is triggered by a fluctuation in EDFA input signal power. It can be observed that while transient of EDFA gain is suppressed, the pump power is adjusted at t = 5ms and t =10ms as the result of this variation.

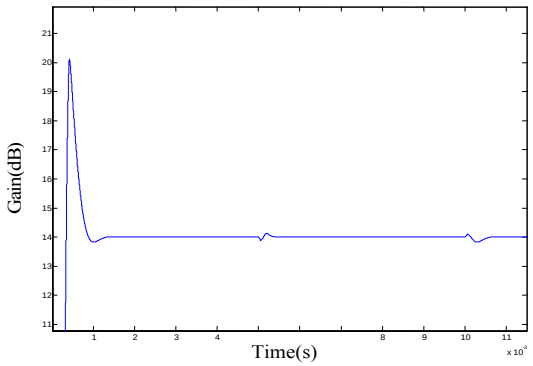

Figure 5. EDFA gain as a function of time for floating-point multiplicative model

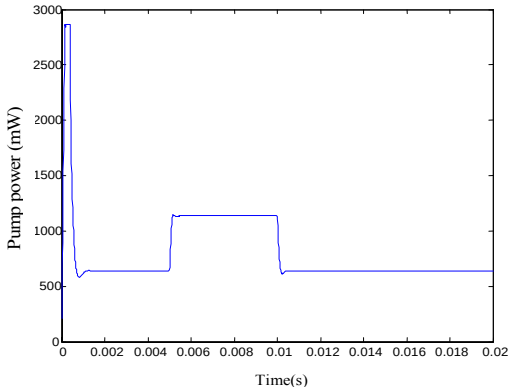

Figure 6. Pump power as a function of time for floating-point multiplicative model

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

The results of fixed-point simulation of EDFA control algorithm is shown in Figure 7 and Figure 8. In Figure 7, it is shown that G_{s0} of 14 dB is maintained and a tolerance of ∂G_s =±0.1 dB is achieved. In Figure 8, it can be observed that the pump power signal is quantised as a result of realising the control algorithm in fixed-point.

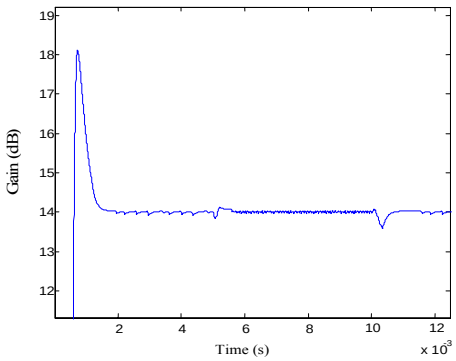

Figure 7. EDFA gain as a function of time for fixed-point multiplicative model

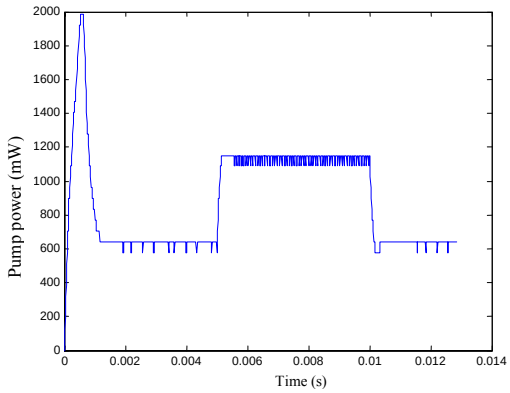

Figure 8. Pump power as a function of time for fixed-point multiplicative model

The results of EDFA emulation are shown in Figure 9 and Figure 10. It is shown that a good agreement exists between these results and those produced using pure software simulation.

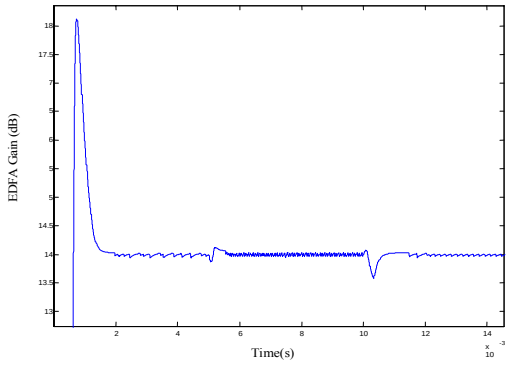

Figure 9. EDFA gain as a function of time, as obtained from system emulation

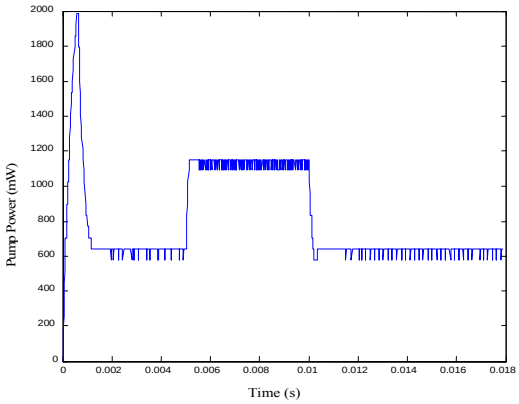

Figure 10. A plot of pump power as a function of time, as obtained from system emulation

5. CONCLUSION

In this paper, it has been shown that the process of realising an algorithm in fixed-point can be facilitated through the numerical analysis of quantisation errors which are propagated in the system. This realisation concept has been implemented as a tool that automates the computation of this error analysis based on the data flow diagram representation of the algorithm.

It was shown that given a generic algorithm consisting of primitive arithmetic operations, the developed framework can provide a fixed-point design which is capable of satisfying the precision constraint of its primary output. Specifically, it has been demonstrated that the technique is viable for the practical implementation of EDFA control algorithms on a dedicated hardware platform. Agreeable outcomes are observed between floating-point simulation, fixed-point simulation, and fixed-point implementation of the control algorithm on Stratix FPGA.

6. REFERENCES

[1] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and Design Environment for DSP ASIC Fixed Point Refinement," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*. Munich, 1999, pp. 271-276.

[2] M. Barberis and N. Shah. (2004). Migrating Signal Processing Applications From Floating-Point to Fixed-Point. *Catalytic Inc. White Paper*. [Online]. Available: http://agilityds.com/literature/catalytic-whitepaper-2.pdf

[3] T. Aamodt and P. Chow, "Embedded ISA support for enhanced floating-point to fixed-point ANSI-C compilation," in *Proc. Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems* San Jose, California, 2000, pp. 128-137.

[4] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A Fixed-Point Design and Simulation Environment," in *Proc. Design Automation and Test in Europe*. Paris, France, 1998, pp. 429-435.

[5] N. Doi, T. Horiyama, M. Nakanishi, and S. Kimura, "Minimization of Fractional Wordlength on Fixed-Point Conversion for High-Level Synthesis," in *Proc. Asia and*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606

South Pacific Design Automation Conference, 2004, pp. 80-85.

[6] K.-I. Kum and W. Sung, "Word-length optimization for high level synthesis of digital signal processing systems," in *Proc. IEEE Int Workshop on Signal Processing Systems*, 1998, pp. 569-578.

[7] J. Yli-Kaakinen and T. Saramaki, "An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength," in *Proc. IEEE Int. Symposium on Circuits and Systems*, vol. 3. Orlando, Florida, 1999, pp. 443-448.

[8] M.-A. Cantin, Y. Blaquiere, Y. Sarvaria, P. Lavoie, and E. Granger, "Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm," in *Proc. IEEE Int. Symposium on Circuits and Systems*, vol. 3. Geneva, 2000, pp. 141-144.

[9] V. J. Mathews and Z. Xie, "Fixed-point error analysis of stochastic gradient adaptive lattice filters," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 38, pp. 70-80, 1990.

[10] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong, "Automatic Floating to Fixed Point Translation and its Application to Post-Rendering 3D Warping," in *Proc. 7th IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa Valley, CA, 1999, pp. 240-248.

[11] S. Kim and W. Sung, "A floating-point to fixed-point assembly program translator for the TMS 320C25," *IEEE Transactions on Circuits and Systems II*, vol. 41, pp. 730-739, 1994.

[12] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Heuristic Datapath Allocation for Multiple Wordlength Systems," in *Proc. Design, Automation and Test in Europe*. Munich, 2001, pp. 791-796.

[13] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimal datapath allocation for multiple-wordlength systems," *Electronic Letters*, vol. 36, pp. 1508-1509, 2000.

[14] W. Sung and K.-I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 43, pp. 3087-3090, 1995.

[15] M. Willems, V. Bursgens, T. Grotker, and H. Meyr, "FRIDGE: an interactive code generation environment for HW/SW codesign," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. 1. Munich, 1997, pp. 287-290.

[16] M. Willems, V. Bursgens, T. Grotker, and H. Meyr, "System level fixed-point design based on interpolative approach," in *Proc. 34th Design Automation Conference*, 1997, pp. 293-298.

[17] K.-I. Kum, J. Kang, and W. Sung, "A floating-point to fixed-point C converter for fixed-point digital signal processors," in *Proc. 2nd SUIF Compiler Workshop*, 1997.

[18] A. A. Gaffar, O. Mencer, W. Luk, and P. Y. K. Cheung, "Unifying Bit-width Optimisation for Fixed-Point and Floating-Point Designs," in *Proc. 12th IEEE Symp. on Field-Programmable Custom Computing Machines*, 2004.

[19] R. Lafore, "Graphs," in *Data Structures & Algorithms in Java*. USA: Mitchell Waite, 1998, pp. 495-533.

[20] G. P. Agrawal, "Optical Amplifiers," in *Fiber-Optic Communication Systems*, K. Chang, Ed., 3rd ed. New York: John Wiley and Sons, 2002, pp. 226-278.

[21] S. V. Kartalopoulos, "Light Amplifiers," in *Introduction to DWDM Technology: Data in a Rainbow*. New York: Wiley - IEEE Press, 2000, pp. 119-130.

[22] *Guide to WDM Technology & Testing : A Unique Reference For the Fiber-Optic Industry*, 2nd ed. Quebec City, Canada: EXFO Electro-Optical Engineering Inc., 2000.

[23] M. Males, "Suppression of transient gain excursions in an erbium-doped fibre amplifier," Ph.D. dissertation, the University of Western Australia, Nedlands, WA, Australia, 2007.

[24] C. Tian and S. Kinoshita, "Analysis and Control of Transient Dynamics of EDFA Pumped by 1480- and 980-nm lasers," *Journal of Lightwave Technology*, vol. 21, pp. 1728-1734, 2003.

[25] J.-P. Laude, *DWDM: Fundamentals, Components, and Applications*. Norwood, M.A.: Artech House Publishers, 2002.

[26] A. K. Srivastava, Y. Sun, J. L. Zyskind, and J. W. Sulhoff, "EDFA transient response to channel loss in WDM transmission system," *IEEE Photonics Technology Letters*, vol. 9, pp. 386-388, 1997.

[27] M. I. Hayee and A. E. Willner, "Transmission Penalties Due to EDFA Gain Transients in Add-Drop Multiplexed WDM Networks," *IEEE Photonics Technology Letters*, vol. 11, pp. 889-891, 1999.

[28] S. Y. Park, H. K. Kim, G. Y. Lyu, S. M. Kang, and S.-Y. Shin, "Dynamic Gain and Output Power Control in a Gain-Flattened Erbium-Doped Fiber Amplifier," *IEEE Photonics Technology Letters*, vol. 10, pp. 787-789, 1998.

[29] Y. Sun, A. K. Srivastava, J. Zhou, and J. W. Sulhoff, "Optical fiber amplifiers for WDM optical networks," *Bell Labs Technical Journal*, vol. 4, pp. 187-206, 2002.

[30] M. Males and A. Cantoni, "Experimental Comparison of Two Pump Control Schemes for Suppressing Transient Gain Excursions in EDFA's," in *Proc. Asia-Pacific Optical Communications*. Shanghai, China, 2005, pp. 6021-48.

[31] M. Males and A. Cantoni, "Stability Analysis of Two Closed-Loop Systems for Suppressing Transient Gain Excursions in an Erbium-Doped Fibre Amplifier," in *Proc. 45th IEEE Conf. on Decision and Control*. San Diego, USA, 2006, pp. 6425-30.

[32] M. Males, A. Cantoni, and J. Tuthill, "Suppression of Transient Gain Excursions in EDFA's: Comparison of Multiplicative and Additive Schemes for Combining Feedforward and Feedback Blocks," in *Proc. Optical Networks and Technologies Conf.* . Pisa, Italy, 2004, pp. 319-326.

[33] A. Bononi and L. A. Rusch, "Doped-Fiber Amplifier Dynamics: A System Perspective," *IEEE Journal of Lightwave Technology*, vol. 16, pp. 945-956, 1998.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5606
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5606