Ambalal V. Patel. International Journal of Engineering Research and Applications
www.ijera.com
ISSN: 2248-9622, Vol. 13, Issue 9, September 2023, pp 32-45

RESEARCH ARTICLE OPEN ACCESS

Some Aspects of Implementation of Soft Transient Free Switching in Embedded Systems

Ambalal V. Patel
Flight Control Laws (CLAW) Directorate,
Aeronautical Development Agency
(Ministry of Defence, Govt. of India),
P.B. 1718, Vimanapura Post, Bangalore – 560017, India

ABSTRACT
The soft faders or Transient Free SWitches (TFSW) are used for gradual transition between signals (instead of instantaneous transition) over a finite time on occurrence of the specific event or setting of a defined discrete. Thus, it helps in eliminating transients in the final outputs and in turn maintaining the safety and performance of the overall system. Implementation of the fader itself involves other levels of considerations like rate of computation, maintaining the memory requirements and overall execution time of the processor to cater for real time computations of the embedded systems of safety critical nature like fly-by-wire flight control system. Various types of proposed TFSWs are available in the literature. This article presents some implementation aspects of the TFSW which include some guidelines for optimizing the implementation in order to deal with execution time and memory requirements. The implementation of the TFSWs with the help of MATLAB *.m file in function forms are provided along with this article. The details are provided here based on the experience gained over a period of time while working on safety critical embedded systems.

Keywords - Fader, Transient Free Switch, data acquisition, algorithms, embedded systems, requirements

Date of Submission: 26-08-2023 Date of acceptance: 07-09-2023

I. INTRODUCTION

Various types of soft elements are used in the embedded systems, especially softwares therein. The ‘soft elements’ here broadly referred to several of the dynamic elements like filters, faders or Transient Free Switches (TFSW), rate limiters etc. are used in the soft form (as part of the computational algorithms within the software) within the embedded systems for various applications [6-7]. The soft faders or transient free switches are used for gradual transition between signals (instead of instantaneous transition) over a finite time on occurrence of the specific event or setting of a defined discrete. Thus, it helps in eliminating the unwanted effects, especially transients in the final outputs or commands and in turn maintaining the safety and performance of the system. Implementation of the fader itself involves other levels of considerations like rate of computation, maintaining the memory requirements and overall execution time of the processor to cater for real time computations of the embedded systems of safety critical nature like fly-by-wire flight control system. There are examples where the implementation of the faders has affected the software functioning and the updates required [6]. Significant efforts have been spent on development of various ‘soft elements’ of embedded systems including their testing and verification [1-5]. Various types of proposed TFSWs and their comparative analysis are detailed in [7].

This article presents some implementation aspects of the TFSW which include some guidelines for optimizing the implementation in order to deal with execution time and memory requirements. The implementation of the TFSWs with the help of MATLAB *.m script file in function forms are provided along with this article. The details are provided here based on the experience gained over a period of time while working on safety critical embedded systems.

The article is organized as given below. After introduction, Section 2 presents brief review of TFSWs including their types. Section 3 presents implementation aspects of TFSW which include optimization of number of independent TFSWs in order to deal with execution time. The section also deals with some aspects of combined event dependent TFSW. These guidelines have been arrived at based on the experiences gained over a period of time while working on the design and development of the safety critical fly-by-wire flight control system. Section 4 deals implementation of the TFSWs in MATLAB *.m function files and relevant results demonstrating their functionality. Section 5 concludes the paper.

II. TRANSIENT FREE SWITCH (TFSW) OR SOFT FADER AND THEIR TYPES

The soft faders or transient free switches are used for gradual transition between signals (instead of instantaneous transition) over a finite time on occurrence of the specific event or setting of defined discrete. Thus, it helps in eliminating the unwanted effects, especially transients in the final outputs or commands and in turn maintaining the safety and performance. Implementation of the fader itself involves other levels of considerations like rate of computation, maintaining the memory requirements and overall execution time of the processor to cater for real time computations of the embedded systems of safety critical nature like fly-by-wire flight control system. Various types of TFSWs as listed below have been proposed in Ref. [7], including the feature of termination of operation at either Less than or Equal to Fader Time (LEFT):

1) Direct Fixed Error Reducing Fader (DFERFD): A direct fixed error (between required and selected output at the instant of Event toggle) per frame (or sampling instant) is reduced from the required signal to reach the desired output smoothly over a specified fader-time.
2) Direct Variable Error Reducing Fader (DVERFD): A direct variable error (between required and selected output at the instant of Event toggle) per frame (or sampling instant) recomputed based on the number of remaining frame counts (or remaining fader-time) is reduced from the required signal to reach the desired output smoothly over a specified fader-time.
3) Scaled Error Reducing Fader (SERFD): A normalized scale factor used for reducing the past signal while increasing the required signal to gradually bring into the selected output over a specified fader-time.

The TFSW of such type of feature can be identified by the nomenclature: TFSW_LEFT, where TFSW could be 'DFERFD', 'DVERFD', 'SERFD'. Thus, they could be identified as 'DFERFD_LEFT', 'DVERFD_LEFT', 'SERFD_LEFT'. The LEFT feature is invoked under the following conditions satisfied together:

1) Fader computation is progressing (event toggle detected and thereafter computations continued) but not completed (before fader time completion which can be found out from the frame counter), and
2) Change in the sign of the error between the past and the present samples is detected. Here error is referred to the difference between the required signal at that instant and selected output of the corresponding past instant.

III. IMPLEMENTATION ASPECTS OF TFSW

With available hardware architecture of the embedded system, the increase in software functionality leads to the constraints on the execution time, i.e., the specific computations of the algorithms should be completed within the specified time frame. In case of safety critical fly-by-wire system for the high-performance combat aircraft, these constraints play a critical role. The onboard software has to play a crucial role of sending the data on the multiple data recording devices while doing the complex and voluminous computations in real time involving several tasks [5]. This section presents a few experiences gained over a period of time and lessons learnt during the design, development and evaluation of fly-by-wire flight control system of high-performance fighter aircraft [1-7]. These examples are based on the black box testing (Hardware in loop testing and analysis):

3.1 Optimization of Number of Independent TFSWs

In the developed on-board real time software has got one independent TFSW for each event at each location in the computational algorithms where reconfiguration takes place. Thus, if there are N number of events, and each event deals with M number of signals at multiple places in the entire application layer (algorithms), then it requires implementation of MxN number of TFSWs. As an example, for N=50 events with fading feature and each event were used at 10 different locations in the computational algorithms for reconfiguration. Then there were total 50*10 = 500 faders had been implemented. It was leading to exceed the execution time. Subsequently, with optimization of some other parts of algorithm implementation and design changes, made to meet the execution time requirements.

It appears that there could be a way to optimize the number TFSWs in the entire set of algorithms / part of the software. Ideally number of TFSWs should be same as that of the number of events / discretes by making the vector of elements of all TFSWs. The number of TFSW should not be increased depending upon the multi-location use. The vector of the elements of TFSW could be a set of inputs (source signals, destination signals), outputs (selected and required signals), intermediate signals (incremental delta etc.) for each event as shown in set of Equations 1 to 3. For the ease of legibility, the symbols used in the Equations have been described therein. Thus, the correspondence between the mathematical symbols used in the Equation and those used in the MATLAB *.m file could be easily established. It would help in dynamically updating these vectors every frame and then distribution of the elements at each location for final faded output computations for each TFSW. The past states of the intermediate parameters (which are required to be stored during TFSW operation is ON) of the TFSWs would reduce from N to 1 for each event. Then it would probably eliminate parts of repetitive computations of TFSWs, and thus it would aid in reducing the memory requirement as well as accelerating the execution time. However, efficacy of such scheme needs to be assessed after actual implementation in the full-fledged application layer in end-to-end manner.

It may be noted that the TFSWs implemented in MATLAB (*.m) files provided along with article has got the feature to deal with the proposed concept of vector of inputs, outputs, and intermediate elements for each specific event, a step towards the ‘Optimization of Number of Independent TFSWs’.

3.2 Combined Event Dependent TFSW

A TFSW gets triggered due to setting of a discrete which in short here could be referred to ‘fader-discrete’. This fader-discrete could be a function of combinations of different events connected with logical AND / OR operations and they have got different fade time. This section presents the aspects of such multiple or combined events / discretes dependent TFSWs, where the events are triggered sequentially or simultaneously.

3.2.1 Sequential Event Trigger Dependent TFSW

A situation wherein one event triggered the TFSW operation and in-between duration (before completing the fader operation), another event triggers the same TFSW, in short during TFSW operation ON, either multiple events or same events keep toggling multiple times. The situation is finally dealt through a toggle of a single fader-discrete. Therefore, the TFSWs described in [7] takes care of such situation. Figure 5 shows the results of such a situation wherein the fader-discrete (Event) is toggled from 1 to 0 in-between when the TFSW operation was ON based on the initial transition from 0 to 1. The results show that the TFSW took care to select appropriate Fader Time and smoothly transit the output to the required signal back.

3.2.2 Parallel or Simultaneous Event Trigger Dependent TFSW

A one TFSW could be triggered due to setting of different events which have got different fade time. If simultaneously such events are toggled then selecting the correct fader time is very important. It could be any of the followings and depends on the design criticality, computational time, and implementation complexities:

1) Maximum of the fader time of the events that are toggled / set simultaneously: It would be simple implementation by keeping a single or few time constants for fader.,
2) Fader time of the event which triggered first during the combined effects. It would be slightly complex than the prior situation (use of maximum fader time). By using the edge trigger or level trigger concept, the status of final fader-discrete along with the selected fader time can be arrived at it.

Anyhow, selection of specific fader time and resultant fader-discrete computation are an external part of the TFSW, and therefore suitable care to be taken for that part of the implementation, separately. The situation is finally dealt through a toggle of a single fader-discrete. Therefore, the TFSWs described in [7] takes care of such situation.

IV. IMPLEMENTATION OF TFSW IN MATLAB *.M FILES AND RESULTS

3.2 Implementation with MATLAB (*.m) files

Figures 1, 2, and 3 present the MATLAB script files (*.m file) in function form for the DFERFD, DVERFD, and SERFD faders, respectively in unified form, i.e., they can used for with and without LEFT feature by setting the input discrete LEFT_DI to True and False, respectively. Detailed description and dimensions of the inputs and outputs signals for each fader / function are given in the beginning of the MATALB function files. User may prepare and provision for the set or vector of inputs and outputs signals in the external interface file (usually referred to driver file), and then use these function-form files like a library module. It may be noted that the TFSWs implemented in MATLAB (*.m) files provided along with article has got the feature to deal with the proposed concept of vector of inputs, outputs, and intermediate elements for each specific event, step towards the 'Optimization of Number of Independent TFSWs'.

3.3 Results

Results of the TFSWs are shown in Figures 4 and 5. These Figures include plots of Event, Required, Selected, Current Signals, and Superimposed Outputs (Selected Signals) of TFSWs separately for with and without LEFT for ease of comparison and understanding. The results here show the functionality of the TFSWs implemented in MATLAB '*.m' file. Figure 6 prominently indicates the efficacy of the functionality of LEFT feature, i.e., feature of termination of operation at either Less than or Equal to Fader Time, if selected signal is reached to the required signal before completion of pre-fixed time of the TFSW.

V. CONCLUSION

Some implementation aspects of the TFSWs including some guidelines for optimizing the implementation in order to deal with execution time and memory requirements are presented in this article. The implementation of the TFSWs with the help of MATLAB *.m script file in function forms are provided along with this article which may be used as library functions. The details are provided here based on the experience gained over a period of time while working on safety critical embedded systems. These guidelines are expected to be applicable for most of the embedded systems and may get enriched further by the experiences and lessons learnt from other systems.

Acknowledgements

The author is grateful to the Aeronautical Development Agency (ADA), Bangalore, India for permitting to publish this work.

REFERENCES

[1] Ambalal V. Patel, Vijay V. Patel, Girish S. Deodhare, and Shyam Chetty, "Clearance of Flight-Control-System Software with Hardware-in-Loop Test Platform", AIAA Journal of Aircraft, Vol. 51, No. 3, May-June 2014, DOI 10.2514/1.C032404.

[2] Ambalal V. Patel, Vijay V. Patel, Girish Deodhare and Shyam Chetty, "Flight Control System clearance using dynamic tests at Hardware-In-Loop Test Platform", Proceedings of International Conference on Avionics Systems (ICAS) 2008, held at RCI, Hyderabad, during February 22-23, 2008.

[3] Guruganesh R., Shyam Chetty, Ambalal V. Patel and Girish Deodhare, "Clearance of LCA Flight Control Laws on Various Ground Test Simulation Platforms", Proceedings of International Conference on Avionics Systems (ICAS) 2008, held at RCI, Hyderabad, during February 22-23, 2008.

[4] Ambalal V. Patel, Vijay V. Patel, Girish Deodhare and Shyam Chetty, "Flight Control System clearance using static tests at Iron Bird", Proceedings AIAA Guidance, Navigation and Control (GNC) conference and exhibit, paper No. 6203 in session No. 31-GNC-14, held at Keystone, Colorado, USA during August 21-24, 2006.

[5] Ambalal V. Patel**,** "Functional Level Data Acquisition Requirement Specification Formulation for Embedded Systems: Challenges, Experiences and Guidelines", International Journal of Engineering Research and Application (IJERA), ISSN: 2248-9622, Vol. 7, Issue 10, (Part -5) October 2017, pp.75-84, DOI: 10.9790/9622-0710057584

[6] Yogananda Jeppu, "Flight Control Software: Mistakes made and Lessons learnt", IEEE Software, PP 67-73, May/June 2013

[7] Ambalal V. Patel, "Various Types of Soft Transient Free Switching in Embedded Systems" International Journal of Engineering Applications and Research, ISSN: 2248-9622, Volume 13, Issue 08, August 2023, PP 01-17.

$$EV = \begin{bmatrix} ev_1 \\ ev_2 \\ \vdots \\ ev_i \\ \vdots \\ ev_n \end{bmatrix}_{(nx1)} ;\ \Lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_i \\ \vdots \\ \lambda_n \end{bmatrix}_{(nx1)} ;\ SC_i = \begin{bmatrix} sc_{i,1} \\ sc_{i,2} \\ \vdots \\ sc_{i,j} \\ \vdots \\ sc_{i,m_i} \end{bmatrix}_{(m_i x1)} ;\ SD_i = \begin{bmatrix} sd_{i,1} \\ sd_{i,2} \\ \vdots \\ sd_{i,j} \\ \vdots \\ sd_{i,m_i} \end{bmatrix}_{(m_i x1)} ;S\,\mathrm{Re}\,q_i = \begin{bmatrix} sreq_{i,1} \\ sreq_{i,2} \\ \vdots \\ sreq_{i,j} \\ \vdots \\ sreq_{i,m_i} \end{bmatrix}_{(m_i x1)}$$

$$Sevt_i = \begin{bmatrix} sevt_{i,1} \\ sevt_{i,2} \\ \vdots \\ sevt_{i,j} \\ \vdots \\ sevt_{i,m_i} \end{bmatrix}_{(m_i x1)} ;\ SCEV = \begin{bmatrix} SC_1 \\ SC_2 \\ \vdots \\ SC_i \\ \vdots \\ SC_n \end{bmatrix}_{\left(\left(\sum\limits_{i=1}^{n} m_i\right)\times 1\right)} ;\ SDEV = \begin{bmatrix} SD_1 \\ SD_2 \\ \vdots \\ SD_i \\ \vdots \\ SD_n \end{bmatrix}_{\left(\left(\sum\limits_{i=1}^{n} m_i\right)\times 1\right)} ;$$

$i = 1, 2, \ldots, n$ for Event number

$j = 1, 2, \ldots, m_i$ for Fader Number of *i*th Event

$n =$ Number of Events or Discretes

$EV =$ Vector of events: ev_i

$\Lambda =$ Vector of Fader Value or Fader Weight for all events: λ_i

$\lambda_i \in [0,\ 1]$ Fader value for *i*th Event (alway between 0 and 1)

$m_i =$ An integer indicating the number of signals for *i*th Fader

$sc_{i,j} =$ Current value of the signal of *j*th Fader of *i*th Event

$sd_{i,j} =$ Destination value of the signal of *j*th Fader of *i*th Event

$SC_i =$ Vector of Current Values of the Signals ($sc_{i,j}$) of all Faders for *i*th Event

$SD_i =$ Vector of Destination Values of the Signals ($sd_{i,j}$) of all Faders for *i*th Event

$S\,\mathrm{Re}\,q_i =$ Vector of Required Values of the Signals ($sreq_{i,j}$) of all Faders for *i*th Event.

They are selected either from any of the SC_i, SD_i *or* in-between depending upon the particular Event status

$Sevt_i =$ Vector of Values of the Signals ($sevt_{i,j}$) at the instant of *i*th Event toggle for all Faders.

$SCEV =$ Vector of Current Values of the Signals ($sc_{i,j}$) of all Faders for all Events

$SDEV =$ Vector of Destination Values of the Signals ($sd_{i,j}$) of all Faders for all Events

$$y_{i,j} = sc_{i,j} \times \lambda_i + sd_{i,j} \times \left(1 - \lambda_i\right)$$

$y_{i,j} =$ Output of the *j*th Fader of *i*th Event

Set of Equations (1)

$$Y_i = \begin{bmatrix} y_{i,1} \\ y_{i,2} \\ \mathrm{M} \\ y_{i,j} \\ \mathrm{M} \\ y_{i,m_i} \end{bmatrix}_{(m_i x1)} = \begin{bmatrix} \begin{bmatrix} sc_{i,1} \\ sc_{i,2} \\ \mathrm{M} \\ sc_{i,j} \\ \mathrm{M} \\ sc_{i,m_i} \end{bmatrix}_{(m_i x1)} \times \lambda_i + \begin{bmatrix} sd_{i,1} \\ sd_{i,2} \\ \mathrm{M} \\ sd_{i,j} \\ \mathrm{M} \\ sd_{i,m_i} \end{bmatrix}_{(m_i x1)} \times (1-\lambda_i) \end{bmatrix}_{(m_i x1)} = SC_i \times \lambda_i + SD_i \times (1-\lambda_i)$$

$$Y_i = SC_i \times \lambda_i + SD_i \times (1-\lambda_i)$$

$$YEV = \begin{bmatrix} Y_1 \\ Y_2 \\ \mathrm{M} \\ Y_i \\ \mathrm{M} \\ Y_n \end{bmatrix}_{\left(\left(\sum_{i=1}^{n} m_i\right) \times 1\right)} = \begin{bmatrix} SC_1 \times \lambda_1 + SD_1 \times (1-\lambda_1) \\ SC_2 \times \lambda_2 + SD_2 \times (1-\lambda_2) \\ \mathrm{M} \\ SC_i \times \lambda_i + SD_i \times (1-\lambda_i) \\ \mathrm{M} \\ SC_n \times \lambda_n + SD_n \times (1-\lambda_n) \end{bmatrix}_{\left(\left(\sum_{i=1}^{n} m_i\right) \times 1\right)} \neq SCEV \times \Lambda + SDEV \times (1-\Lambda)$$

YEV is a column vector of size (Rows x Columns) = R_{YEV} *x* C_{YEV} = $\left(\sum_{i=1}^{n} m_i\right) \times 1$

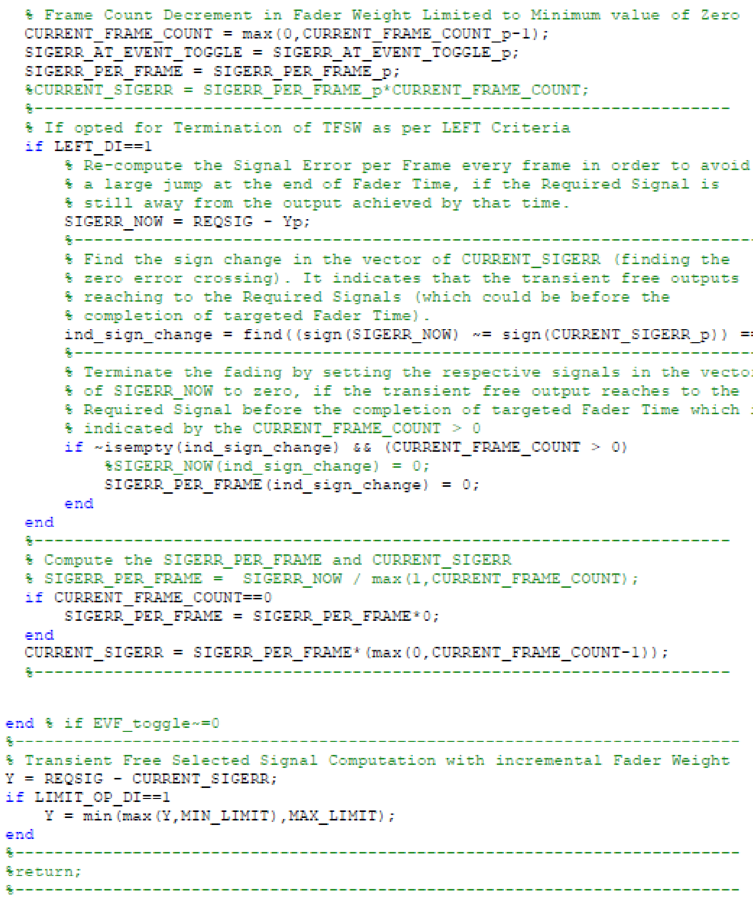
Where $R_{YEV} = \sum_{i=1}^{n} m_i$ and $C_{YEV} = 1$

N_f = Number of Faders

Thus, ideally the numbers of faders required to implemented should not be more than the number of Events used. If some events are not used for any fading of the signals while reconfiguring the computations, then the number of faders required would be less than the number of Events used. This, at the most Number of Faders (N_f) = Number of Unique Events or Discretes used (*n*).

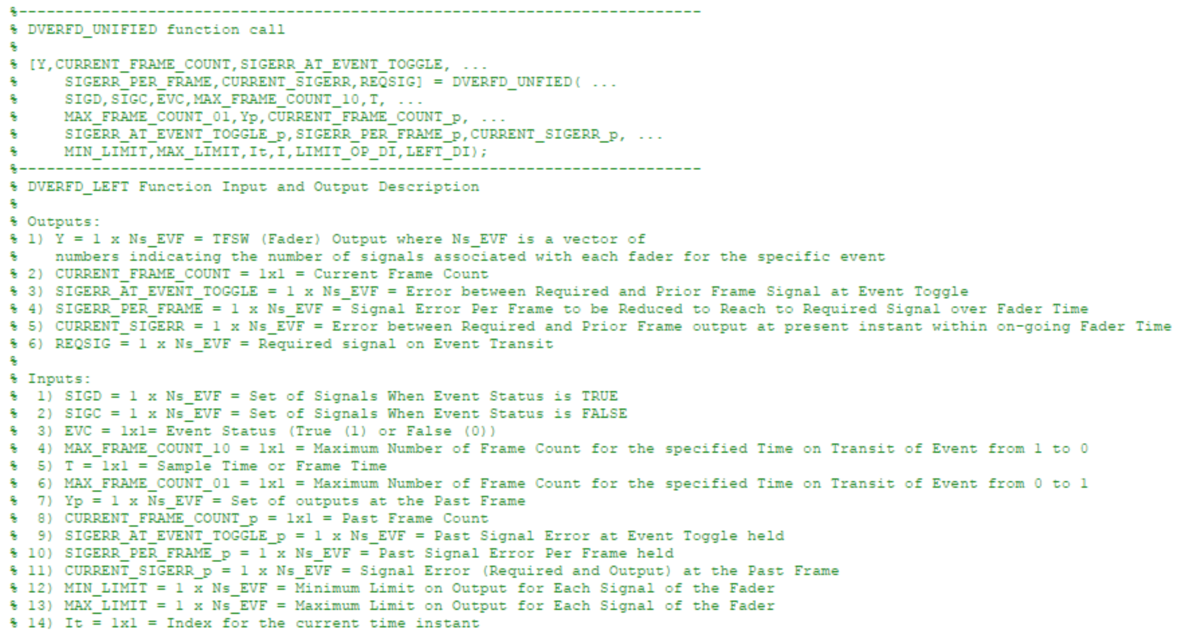
$$YEV = \begin{bmatrix} Y_1 \\ Y_2 \\ \mathrm{M} \\ Y_i \\ \mathrm{M} \\ Y_n \end{bmatrix}_{\left(\left(\sum\limits_{i=1}^{n} m_i\right)\times 1\right)} = \begin{bmatrix} SC_1 \times \lambda_1 + SD_1 \times \left(1-\lambda_1\right) \\ SC_2 \times \lambda_2 + SD_2 \times \left(1-\lambda_2\right) \\ \mathrm{M} \\ SC_i \times \lambda_i + SD_i \times \left(1-\lambda_i\right) \\ \mathrm{M} \\ SC_n \times \lambda_n + SD_n \times \left(1-\lambda_n\right) \end{bmatrix}_{\left(\left(\sum\limits_{i=1}^{n} m_i\right)\times 1\right)} = \begin{bmatrix} \begin{bmatrix} y_{1,1} \\ y_{1,2} \\ \mathrm{M} \\ y_{1,j} \\ \mathrm{M} \\ y_{1,m_1} \end{bmatrix} \\ \begin{bmatrix} y_{2,1} \\ y_{2,2} \\ \mathrm{M} \\ y_{2,j} \\ \mathrm{M} \\ y_{2,m_2} \end{bmatrix} \\ \mathrm{M} \\ \begin{bmatrix} y_{i,1} \\ y_{i,2} \\ \mathrm{M} \\ y_{i,j} \\ \mathrm{M} \\ y_{i,m_i} \end{bmatrix} \\ \mathrm{M} \\ \begin{bmatrix} y_{n,1} \\ y_{n,2} \\ \mathrm{M} \\ y_{n,j} \\ \mathrm{M} \\ y_{n,m_n} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \left[\begin{bmatrix} sc_{1,1} \\ sc_{1,2} \\ \mathrm{M} \\ sc_{1,j} \\ \mathrm{M} \\ sc_{1,m_1} \end{bmatrix} \times \lambda_1 + \begin{bmatrix} sd_{1,1} \\ sd_{1,2} \\ \mathrm{M} \\ sd_{1,j} \\ \mathrm{M} \\ sd_{1,m_1} \end{bmatrix} \times \left(1-\lambda_1\right) \right] \\ \left[\begin{bmatrix} sc_{2,1} \\ sc_{2,2} \\ \mathrm{M} \\ sc_{2,j} \\ \mathrm{M} \\ sc_{2,m_2} \end{bmatrix} \times \lambda_2 + \begin{bmatrix} sd_{2,1} \\ sd_{2,2} \\ \mathrm{M} \\ sd_{2,j} \\ \mathrm{M} \\ sd_{2,m_2} \end{bmatrix} \times \left(1-\lambda_2\right) \right] \\ \mathrm{M} \\ \left[\begin{bmatrix} sc_{i,1} \\ sc_{i,2} \\ \mathrm{M} \\ sc_{i,j} \\ \mathrm{M} \\ sc_{i,m_i} \end{bmatrix} \times \lambda_i + \begin{bmatrix} sd_{i,1} \\ sd_{i,2} \\ \mathrm{M} \\ sd_{i,j} \\ \mathrm{M} \\ sd_{i,m_i} \end{bmatrix} \times \left(1-\lambda_i\right) \right] \\ \mathrm{M} \\ \left[\begin{bmatrix} sc_{n,1} \\ sc_{n,2} \\ \mathrm{M} \\ sc_{n,j} \\ \mathrm{M} \\ sc_{n,m_n} \end{bmatrix} \times \lambda_n + \begin{bmatrix} sd_{n,1} \\ sd_{n,2} \\ \mathrm{M} \\ sd_{n,j} \\ \mathrm{M} \\ sd_{n,m_n} \end{bmatrix} \times \left(1-\lambda_n\right) \right] \end{bmatrix}_{\left(\left(\sum\limits_{i=1}^{n} m_i\right)\times 1\right)}$$

```
%-------------------------------------------------------------------------
% DFERFD_UNIFIED function call
%
% function [Y,CURRENT_FRAME_COUNT, SIGERR_AT_EVENT_TOGGLE, ...
% SIGERR_PER_FRAME,CURRENT_SIGERR,REQSIG] = DFERFD_UNIFIED( ..
% SIGD,SIGC,EVC,MAX_FRAME_COUNT_10,T, ...
% MAX_FRAME_COUNT_01,Yp,CURRENT_FRAME_COUNT_p, ...
% SIGERR_AT_EVENT_TOGGLE_p,SIGERR_PER_FRAME_p,CURRENT_SIGERR_p, ...
% MIN_LIMIT,MAX_LIMIT,It,I,LIMIT_OP_DI,LEFT_DI);
%-------------------------------------------------------------------------
% DFERFD_UNIFIED Function Input and Output Description
%
% Outputs:
% 1) Y = 1 x Ns_EVF = TFSW (Fader) Output where Ns_EVF is a vector of
%    numbers indicating the number of signals associated with each fader for the specific event
% 2) CURRENT_FRAME_COUNT = 1x1 = Current Frame Count
% 3) SIGERR_AT_EVENT_TOGGLE = 1 x Ns_EVF = Error between Required and Prior Frame Signal at Event Toggle
% 4) SIGERR_PER_FRAME = 1 x Ns_EVF = Signal Error Per Frame to be Reduced to Reach to Required Signal over Fader Time
% 5) CURRENT_SIGERR = 1 x Ns_EVF = Error between Required and Prior Frame output at present instant within on-going Fader Time
% 6) REQSIG = 1 x Ns_EVF = Required signal on Event Transit
%
% Inputs:
%  1) SIGD = 1 x Ns_EVF = Set of Signals When Event Status is TRUE
%  2) SIGC = 1 x Ns_EVF = Set of Signals When Event Status is FALSE
%  3) EVC = 1x1= Event Status (True (1) or False (0))
%  4) MAX_FRAME_COUNT_10 = 1x1 = Maximum Number of Frame Count for the specified Time on Transit of Event from 1 to 0
%  5) T = 1x1 = Sample Time or Frame Time
%  6) MAX_FRAME_COUNT_01 = 1x1 = Maximum Number of Frame Count for the specified Time on Transit of Event from 0 to 1
%  7) Yp = 1 x Ns_EVF = Set of outputs at the Past Frame
%  8) CURRENT_FRAME_COUNT_p = 1x1 = Past Frame Count
%  9) SIGERR_AT_EVENT_TOGGLE_p = 1 x Ns_EVF = Past Signal Error at Event Toggle held
% 10) SIGERR_PER_FRAME_p = 1 x Ns_EVF = Past Signal Error Per Frame held
% 11) CURRENT_SIGERR_p = 1 x Ns_EVF = Signal Error (Required and Output) at the Past Frame
% 12) MIN_LIMIT = 1 x Ns_EVF = Minimum Limit on Output for Each Signal of the Fader
% 13) MAX_LIMIT = 1 x Ns_EVF = Maximum Limit on Output for Each Signal of the Fader
% 14) It = 1x1 = Index for the current time instant

% 15) I = 1x1 = Index for the current Event
% 16) LIMIT_OP_DI = 1x1 = Discrete for Limit (1) / Do not Limit (0) output
% 17) LEFT_DI = 1x1 = Discrete for Termination of TFSW Operation if 'Less
%     Than OR Equal to Fader Time' (LEFT) Criteria satisfied
%     (True(1): Opted; False(0): Not Opted)
%-------------------------------------------------------------------------
function [Y,CURRENT_FRAME_COUNT, SIGERR_AT_EVENT_TOGGLE, ...
    SIGERR_PER_FRAME,CURRENT_SIGERR,REQSIG] = DFERFD_UNIFIED( ...
    SIGD,SIGC,EVC,MAX_FRAME_COUNT_10,T, ...
    MAX_FRAME_COUNT_01,Yp,CURRENT_FRAME_COUNT_p, ...
    SIGERR_AT_EVENT_TOGGLE_p,SIGERR_PER_FRAME_p,CURRENT_SIGERR_p, ...
    MIN_LIMIT,MAX_LIMIT,It,I,LIMIT_OP_DI,LEFT_DI)
%-------------------------------------------------------------------------
% Find Status of Event Toggle including initialization
EVF_toggle = EVC(It)-EVC(max((It-1), 1));
%-------------------------------------------------------------------------
if EVC(It)==0
    REQSIG = SIGC(It,:);
elseif EVC(It)==1
    REQSIG = SIGD(It,:);
end
%-------------------------------------------------------------------------
% Fader weight computation on Event Toggle
if EVF_toggle~=0
    %---------------------------------------------------------------------
    if EVC(It)==1
        MAX_FRAME_COUNT = MAX_FRAME_COUNT_01(I);
    elseif EVC(It)==0
        MAX_FRAME_COUNT = MAX_FRAME_COUNT_10(I);
    end
    CURRENT_FRAME_COUNT = MAX_FRAME_COUNT;
    SIGERR_AT_EVENT_TOGGLE = REQSIG - Yp;
    SIGERR_PER_FRAME =  SIGERR_AT_EVENT_TOGGLE / MAX_FRAME_COUNT;
    CURRENT_SIGERR = SIGERR_PER_FRAME*(max(0,(CURRENT_FRAME_COUNT-1)));
    %---------------------------------------------------------------------
elseif EVF_toggle==0
    %---------------------------------------------------------------------
```

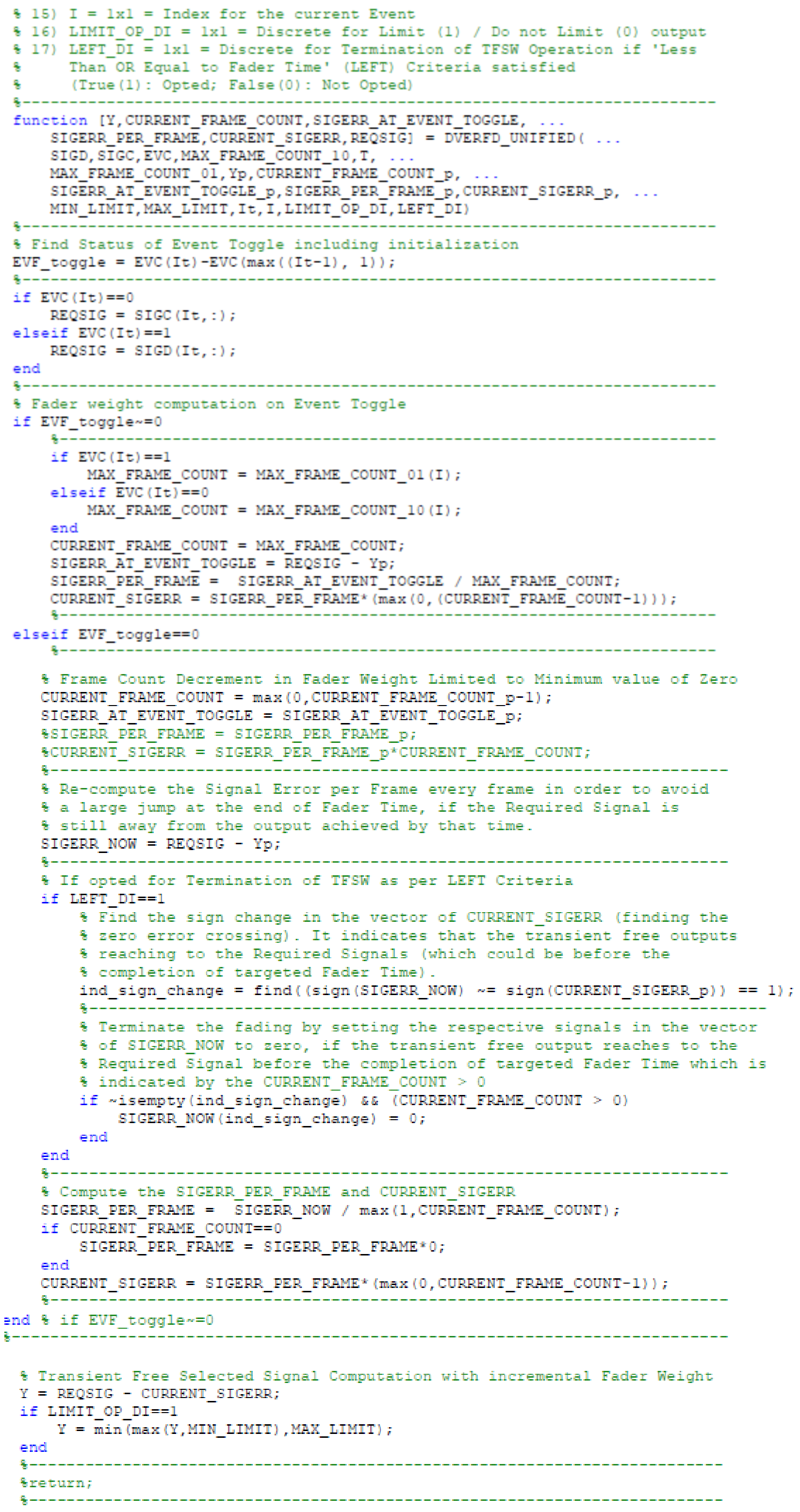


```
  % Frame Count Decrement in Fader Weight Limited to Minimum value of Zero
  CURRENT_FRAME_COUNT = max(0,CURRENT_FRAME_COUNT_p-1);
  SIGERR_AT_EVENT_TOGGLE = SIGERR_AT_EVENT_TOGGLE_p;
  SIGERR_PER_FRAME = SIGERR_PER_FRAME_p;
  %CURRENT_SIGERR = SIGERR_PER_FRAME_p*CURRENT_FRAME_COUNT;
  %-----------------------------------------------------------------------
  % If opted for Termination of TFSW as per LEFT Criteria
  if LEFT_DI==1
      % Re-compute the Signal Error per Frame every frame in order to avoid
      % a large jump at the end of Fader Time, if the Required Signal is
      % still away from the output achieved by that time.
      SIGERR_NOW = REQSIG - Yp;
      %-------------------------------------------------------------------
      % Find the sign change in the vector of CURRENT_SIGERR (finding the
      % zero error crossing). It indicates that the transient free outputs
      % reaching to the Required Signals (which could be before the
      % completion of targeted Fader Time).
      ind_sign_change = find((sign(SIGERR_NOW) ~= sign(CURRENT_SIGERR_p)) == 1);
      %-------------------------------------------------------------------
      % Terminate the fading by setting the respective signals in the vector
      % of SIGERR_NOW to zero, if the transient free output reaches to the
      % Required Signal before the completion of targeted Fader Time which is
      % indicated by the CURRENT_FRAME_COUNT > 0
      if ~isempty(ind_sign_change) && (CURRENT_FRAME_COUNT > 0)
          %SIGERR_NOW(ind_sign_change) = 0;
          SIGERR_PER_FRAME(ind_sign_change) = 0;
      end
  end
  %-----------------------------------------------------------------------
  % Compute the SIGERR_PER_FRAME and CURRENT_SIGERR
  % SIGERR_PER_FRAME =  SIGERR_NOW / max(1,CURRENT_FRAME_COUNT);
  if CURRENT_FRAME_COUNT==0
      SIGERR_PER_FRAME = SIGERR_PER_FRAME*0;
  end
  CURRENT_SIGERR = SIGERR_PER_FRAME*(max(0,CURRENT_FRAME_COUNT-1));
  %-----------------------------------------------------------------------


end % if EVF_toggle~=0
%-----------------------------------------------------------------------
% Transient Free Selected Signal Computation with incremental Fader Weight
Y = REQSIG - CURRENT_SIGERR;
if LIMIT_OP_DI==1
    Y = min(max(Y,MIN_LIMIT),MAX_LIMIT);
end
%-----------------------------------------------------------------------
%return;
%-----------------------------------------------------------------------
```

Figure 1: DFERFD_UNIFIED TFSW function *.m file (Direct Fixed Error Reducing Fader)



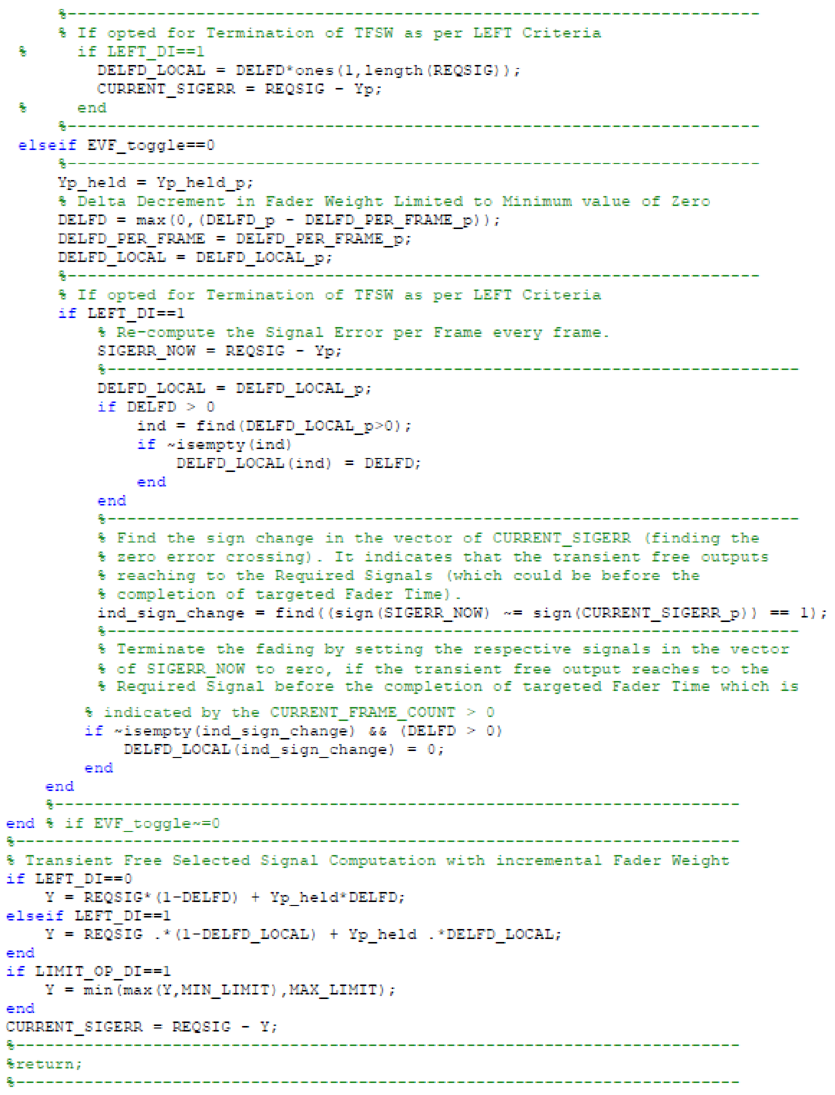
```
%-----------------------------------------------------------------------
% DVERFD_UNIFIED function call
%
% [Y,CURRENT_FRAME_COUNT,SIGERR_AT_EVENT_TOGGLE, ...
%     SIGERR_PER_FRAME,CURRENT_SIGERR,REQSIG] = DVERFD_UNFIED( ...
%     SIGD,SIGC,EVC,MAX_FRAME_COUNT_10,T, ...
%     MAX_FRAME_COUNT_01,Yp,CURRENT_FRAME_COUNT_p, ...
%     SIGERR_AT_EVENT_TOGGLE_p,SIGERR_PER_FRAME_p,CURRENT_SIGERR_p, ...
%     MIN_LIMIT,MAX_LIMIT,It,I,LIMIT_OP_DI,LEFT_DI);
%-----------------------------------------------------------------------
% DVERFD_LEFT Function Input and Output Description
%
% Outputs:
% 1) Y = 1 x Ns_EVF = TFSW (Fader) Output where Ns_EVF is a vector of
%    numbers indicating the number of signals associated with each fader for the specific event
% 2) CURRENT_FRAME_COUNT = 1x1 = Current Frame Count
% 3) SIGERR_AT_EVENT_TOGGLE = 1 x Ns_EVF = Error between Required and Prior Frame Signal at Event Toggle
% 4) SIGERR_PER_FRAME = 1 x Ns_EVF = Signal Error Per Frame to be Reduced to Reach to Required Signal over Fader Time
% 5) CURRENT_SIGERR = 1 x Ns_EVF = Error between Required and Prior Frame output at present instant within on-going Fader Time
% 6) REQSIG = 1 x Ns_EVF = Required signal on Event Transit
%
% Inputs:
%  1) SIGD = 1 x Ns_EVF = Set of Signals When Event Status is TRUE
%  2) SIGC = 1 x Ns_EVF = Set of Signals When Event Status is FALSE
%  3) EVC = 1x1= Event Status (True (1) or False (0))
%  4) MAX_FRAME_COUNT_10 = 1x1 = Maximum Number of Frame Count for the specified Time on Transit of Event from 1 to 0
%  5) T = 1x1 = Sample Time or Frame Time
%  6) MAX_FRAME_COUNT_01 = 1x1 = Maximum Number of Frame Count for the specified Time on Transit of Event from 0 to 1
%  7) Yp = 1 x Ns_EVF = Set of outputs at the Past Frame
%  8) CURRENT_FRAME_COUNT_p = 1x1 = Past Frame Count
%  9) SIGERR_AT_EVENT_TOGGLE_p = 1 x Ns_EVF = Past Signal Error at Event Toggle held
% 10) SIGERR_PER_FRAME_p = 1 x Ns_EVF = Past Signal Error Per Frame held
% 11) CURRENT_SIGERR_p = 1 x Ns_EVF = Signal Error (Required and Output) at the Past Frame
% 12) MIN_LIMIT = 1 x Ns_EVF = Minimum Limit on Output for Each Signal of the Fader
% 13) MAX_LIMIT = 1 x Ns_EVF = Maximum Limit on Output for Each Signal of the Fader
% 14) It = 1x1 = Index for the current time instant
```



```
% 15) I = 1x1 = Index for the current Event
% 16) LIMIT_OP_DI = 1x1 = Discrete for Limit (1) / Do not Limit (0) output
% 17) LEFT_DI = 1x1 = Discrete for Termination of TFSW Operation if 'Less
%     Than OR Equal to Fader Time' (LEFT) Criteria satisfied
%     (True(1): Opted; False(0): Not Opted)
%-------------------------------------------------------------------------
function [Y,CURRENT_FRAME_COUNT,SIGERR_AT_EVENT_TOGGLE, ...
    SIGERR_PER_FRAME,CURRENT_SIGERR,REQSIG] = DVERFD_UNIFIED( ...
    SIGD,SIGC,EVC,MAX_FRAME_COUNT_10,T, ...
    MAX_FRAME_COUNT_01,Yp,CURRENT_FRAME_COUNT_p, ...
    SIGERR_AT_EVENT_TOGGLE_p,SIGERR_PER_FRAME_p,CURRENT_SIGERR_p, ...
    MIN_LIMIT,MAX_LIMIT,It,I,LIMIT_OP_DI,LEFT_DI)
%-------------------------------------------------------------------------
% Find Status of Event Toggle including initialization
EVF_toggle = EVC(It)-EVC(max((It-1), 1));
%-------------------------------------------------------------------------
if EVC(It)==0
    REQSIG = SIGC(It,:);
elseif EVC(It)==1
    REQSIG = SIGD(It,:);
end
%-------------------------------------------------------------------------
% Fader weight computation on Event Toggle
if EVF_toggle~=0
    %---------------------------------------------------------------------
    if EVC(It)==1
        MAX_FRAME_COUNT = MAX_FRAME_COUNT_01(I);
    elseif EVC(It)==0
        MAX_FRAME_COUNT = MAX_FRAME_COUNT_10(I);
    end
    CURRENT_FRAME_COUNT = MAX_FRAME_COUNT;
    SIGERR_AT_EVENT_TOGGLE = REQSIG - Yp;
    SIGERR_PER_FRAME =  SIGERR_AT_EVENT_TOGGLE / MAX_FRAME_COUNT;
    CURRENT_SIGERR = SIGERR_PER_FRAME*(max(0,(CURRENT_FRAME_COUNT-1)));
    %---------------------------------------------------------------------
elseif EVF_toggle==0
    %---------------------------------------------------------------------

   % Frame Count Decrement in Fader Weight Limited to Minimum value of Zero
   CURRENT_FRAME_COUNT = max(0,CURRENT_FRAME_COUNT_p-1);
   SIGERR_AT_EVENT_TOGGLE = SIGERR_AT_EVENT_TOGGLE_p;
   %SIGERR_PER_FRAME = SIGERR_PER_FRAME_p;
   %CURRENT_SIGERR = SIGERR_PER_FRAME_p*CURRENT_FRAME_COUNT;
   %-----------------------------------------------------------------------
   % Re-compute the Signal Error per Frame every frame in order to avoid
   % a large jump at the end of Fader Time, if the Required Signal is
   % still away from the output achieved by that time.
   SIGERR_NOW = REQSIG - Yp;
   %-----------------------------------------------------------------------
   % If opted for Termination of TFSW as per LEFT Criteria
   if LEFT_DI==1
       % Find the sign change in the vector of CURRENT_SIGERR (finding the
       % zero error crossing). It indicates that the transient free outputs
       % reaching to the Required Signals (which could be before the
       % completion of targeted Fader Time).
       ind_sign_change = find((sign(SIGERR_NOW) ~= sign(CURRENT_SIGERR_p)) == 1);
       %-----------------------------------------------------------------------
       % Terminate the fading by setting the respective signals in the vector
       % of SIGERR_NOW to zero, if the transient free output reaches to the
       % Required Signal before the completion of targeted Fader Time which is
       % indicated by the CURRENT_FRAME_COUNT > 0
       if ~isempty(ind_sign_change) && (CURRENT_FRAME_COUNT > 0)
           SIGERR_NOW(ind_sign_change) = 0;
       end
   end
   %-----------------------------------------------------------------------
   % Compute the SIGERR_PER_FRAME and CURRENT_SIGERR
   SIGERR_PER_FRAME =  SIGERR_NOW / max(1,CURRENT_FRAME_COUNT);
   if CURRENT_FRAME_COUNT==0
       SIGERR_PER_FRAME = SIGERR_PER_FRAME*0;
   end
   CURRENT_SIGERR = SIGERR_PER_FRAME*(max(0,CURRENT_FRAME_COUNT-1));
   %-----------------------------------------------------------------------
end % if EVF_toggle~=0
%--------------------------------------------------------------------------

  % Transient Free Selected Signal Computation with incremental Fader Weight
  Y = REQSIG - CURRENT_SIGERR;
  if LIMIT_OP_DI==1
      Y = min(max(Y,MIN_LIMIT),MAX_LIMIT);
  end
  %-----------------------------------------------------------------------
  %return;
  %-----------------------------------------------------------------------
```

Figure 2: DVERFD_UNIFIED TFSW function *.m file (Direct Variable Error Reducing Fader)

```
%-----------------------------------------------------------------------
% SERFD_UNIFIED function call
%
% function [Y,Yp_held,DELFD,DELFD_PER_FRAME, ...
% REQSIG,CURRENT_SIGERR,DELFD_LOCAL] = SERFD_UNIFIED( ...
% SIGD,SIGC,EVC,MAX_DELFD_PER_FRAME_EVT_10,T, ...
% MAX_DELFD_PER_FRAME_EVT_01,Yp,Yp_held_p, ...
% DELFD_PER_FRAME_p,DELFD_p,MIN_LIMIT,MAX_LIMIT,It,I, ...
% CURRENT_SIGERR_p,DELFD_LOCAL_p,LIMIT_OP_DI,LEFT_DI);
%-----------------------------------------------------------------------
% SERFD_LEFT Function Input and Output Description
%
% Outputs:
% 1) Y = 1 x Ns_EVF = TFSW (Fader) Output where Ns_EVF is a vector of
%    numbers indicating the number of signals associated with each fader for the specific event
% 2) Yp_held = 1 x Ns_EVF = Past output at the instant of Event Toggle held
% 3) DELFD = 1x1 = Normalized Fader Weight at current frame
% 4) DELFD_PER_FRAME = 1x1 = Normalized Fader Weight per Frame to be Reduced to Reach to Required Signal over Fader Time
% 5) REQSIG = 1 x Ns_EVF = Required signal on Event Transit
% 6) CURRENT_SIGERR = 1 x Ns_EVF = Error between Required and Prior Frame output at present instant within on-going Fader Time
% 7) DELFD_LOCAL = 1 x Ns_EVF = Set of Normalized Fader Weights at current frame for internal computations
%
% Inputs:
%  1) SIGD = 1 x Ns_EVF = Set of Signals When Event Status is TRUE
%  2) SIGC = 1 x Ns_EVF = Set of Signals When Event Status is FALSE
%  3) EVC = 1x1= Event Status (True (1) or False (0))
%  4) MAX_DELFD_PER_FRAME_EVT_10 = 1x1 = Maximum Normalized Weight Per Frame for the specified Time on Transit of Event from 1
to 0
%  5) T = 1x1 = Sample Time or Frame Time
%  6) MAX_DELFD_PER_FRAME_EVT_01 = 1x1 = Maximum Normalized Weight Per Frame for the specified Time on Transit of Event from 0
to 1
%  7) Yp = 1 x Ns_EVF = Set of outputs at the Past Frame
%  8) Yp_held_p = 1 x Ns_EVF = Past output at the instant of Event Toggle held
%  9) DELFD_PER_FRAME_p = 1x1 = Past Frame Normalized Fader Weight per Frame to be Reduced to Reach to Required Signal over
Fader Time
% 10) DELFD_p = 1x1 = Past Frame Normalized Fader Weight

% 11) MIN_LIMIT = 1 x Ns_EVF = Minimum Limit on Output for Each Signal of the Fader
% 12) MAX_LIMIT = 1 x Ns_EVF = Maximum Limit on Output for Each Signal of the Fader
% 13) It = 1x1 = Index for the current time instant
% 14) I = 1x1 = Index for the current Event
% 15) CURRENT_SIGERR_p = 1 x Ns_EVF = Signal Error (Required and Output) at the Past Frame
% 16) DELFD_LOCAL_p = 1 x Ns_EVF = Set of Normalized Fader Weights at Past frame for internal computations
% 17) LIMIT_OP_DI = 1x1 = Discrete for Limit (1) / Do not Limit (0) output
% 18) LEFT_DI = 1x1 = Discrete for Termination of TFSW Operation if 'Less
%     Than OR Equal to Fader Time' (LEFT) Criteria satisfied
%     (True(1): Opted; False(0): Not Opted)
%-----------------------------------------------------------------------
function [Y,Yp_held,DELFD,DELFD_PER_FRAME, ...
    REQSIG,CURRENT_SIGERR,DELFD_LOCAL] = SERFD_UNIFIED( ...
    SIGD,SIGC,EVC,MAX_DELFD_PER_FRAME_EVT_10,T, ...
    MAX_DELFD_PER_FRAME_EVT_01,Yp,Yp_held_p, ...
    DELFD_PER_FRAME_p,DELFD_p,MIN_LIMIT,MAX_LIMIT,It,I, ...
    CURRENT_SIGERR_p,DELFD_LOCAL_p,LIMIT_OP_DI,LEFT_DI)
%-----------------------------------------------------------------------
% Find Status of Event Toggle including initialization
EVF_toggle = EVC(It)-EVC(max((It-1), 1));
%-----------------------------------------------------------------------
if EVC(It)==0
    REQSIG = SIGC(It,:);
elseif EVC(It)==1
    REQSIG = SIGD(It,:);
end
%-----------------------------------------------------------------------
% Fader weight computation on Event Toggle
if EVF_toggle~=0
    %-------------------------------------------------------------------
    Yp_held = Yp;
    DELFD = 1;
    if EVC(It)==1
        DELFD_PER_FRAME = MAX_DELFD_PER_FRAME_EVT_01(I);
    elseif EVC(It)==0
        DELFD_PER_FRAME = MAX_DELFD_PER_FRAME_EVT_10(I);
    end
```



```
     %----------------------------------------------------------------------
     % If opted for Termination of TFSW as per LEFT Criteria
 %     if LEFT_DI==1
         DELFD_LOCAL = DELFD*ones(1,length(REQSIG));
         CURRENT_SIGERR = REQSIG - Yp;
 %     end
     %----------------------------------------------------------------------
 elseif EVF_toggle==0
     %----------------------------------------------------------------------
     Yp_held = Yp_held_p;
     % Delta Decrement in Fader Weight Limited to Minimum value of Zero
     DELFD = max(0,(DELFD_p - DELFD_PER_FRAME_p));
     DELFD_PER_FRAME = DELFD_PER_FRAME_p;
     DELFD_LOCAL = DELFD_LOCAL_p;
     %----------------------------------------------------------------------
     % If opted for Termination of TFSW as per LEFT Criteria
     if LEFT_DI==1
         % Re-compute the Signal Error per Frame every frame.
         SIGERR_NOW = REQSIG - Yp;
         %----------------------------------------------------------------------
         DELFD_LOCAL = DELFD_LOCAL_p;
         if DELFD > 0
             ind = find(DELFD_LOCAL_p>0);
             if ~isempty(ind)
                 DELFD_LOCAL(ind) = DELFD;
             end
         end
         %----------------------------------------------------------------------
         % Find the sign change in the vector of CURRENT_SIGERR (finding the
         % zero error crossing). It indicates that the transient free outputs
         % reaching to the Required Signals (which could be before the
         % completion of targeted Fader Time).
         ind_sign_change = find((sign(SIGERR_NOW) ~= sign(CURRENT_SIGERR_p)) == 1);
         %----------------------------------------------------------------------
         % Terminate the fading by setting the respective signals in the vector
         % of SIGERR_NOW to zero, if the transient free output reaches to the
         % Required Signal before the completion of targeted Fader Time which is
        % indicated by the CURRENT_FRAME_COUNT > 0
        if ~isempty(ind_sign_change) && (DELFD > 0)
            DELFD_LOCAL(ind_sign_change) = 0;
        end
    end
    %----------------------------------------------------------------------
end % if EVF_toggle~=0
%----------------------------------------------------------------------
% Transient Free Selected Signal Computation with incremental Fader Weight
if LEFT_DI==0
    Y = REQSIG*(1-DELFD) + Yp_held*DELFD;
elseif LEFT_DI==1
    Y = REQSIG .*(1-DELFD_LOCAL) + Yp_held .*DELFD_LOCAL;
end
if LIMIT_OP_DI==1
    Y = min(max(Y,MIN_LIMIT),MAX_LIMIT);
end
CURRENT_SIGERR = REQSIG - Y;
%----------------------------------------------------------------------
%return;
%----------------------------------------------------------------------
```

Figure 3: SERFD_UNIFIED TFSW function *.m file (Scaled Error Reducing Fader)

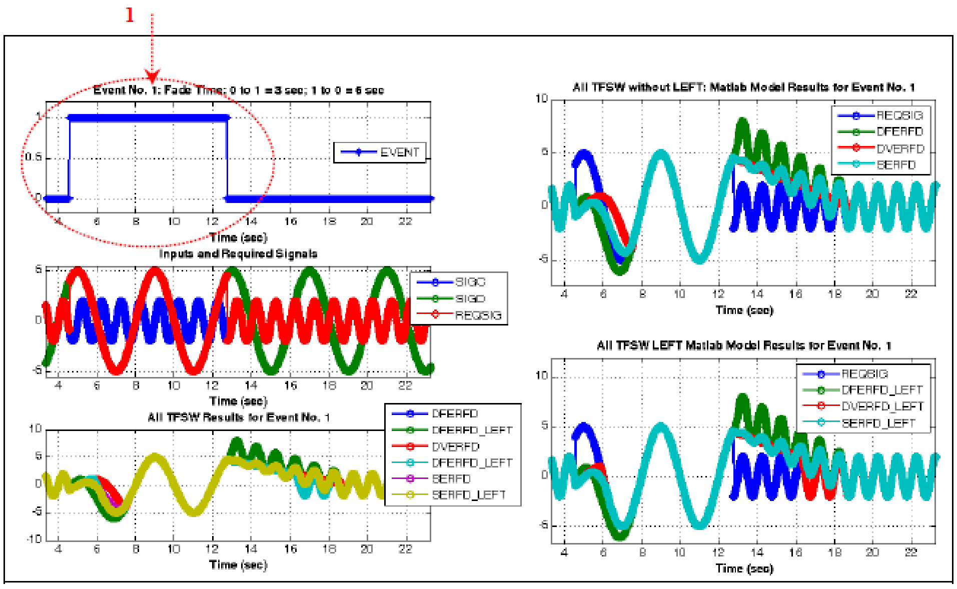


Figure 4: ALL TFSW Results of Matlab *.m file model, With and Without LEFT (Unified TFSW) at Event Toggle Segment No. 1.

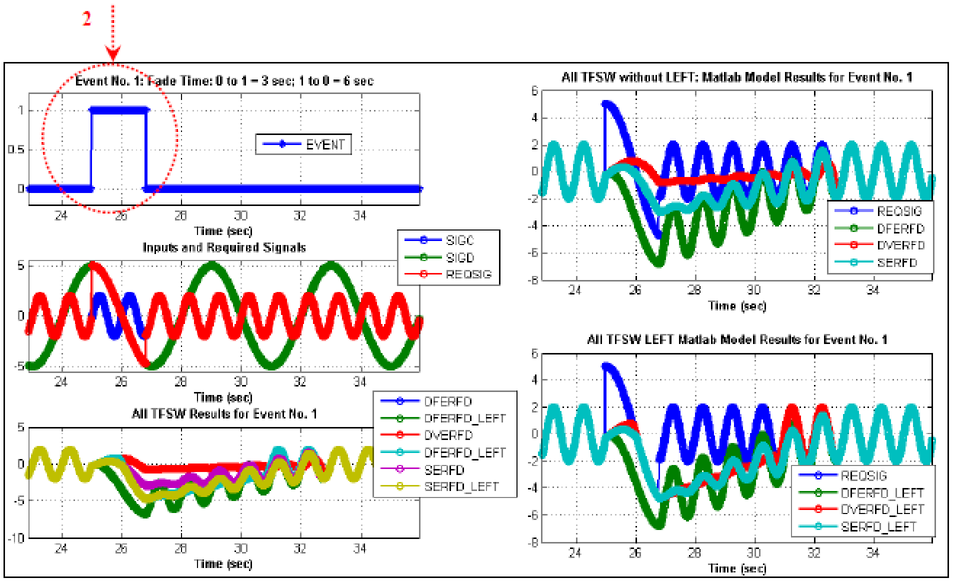


Figure 6: ALL TFSW Results, With and Without LEFT: Event Toggle Segment No. 2. Operation of the TFSW upon toggle of the Event from 1 to 0 when TFSW operation was ON based on the prior transition from 0 to 1.

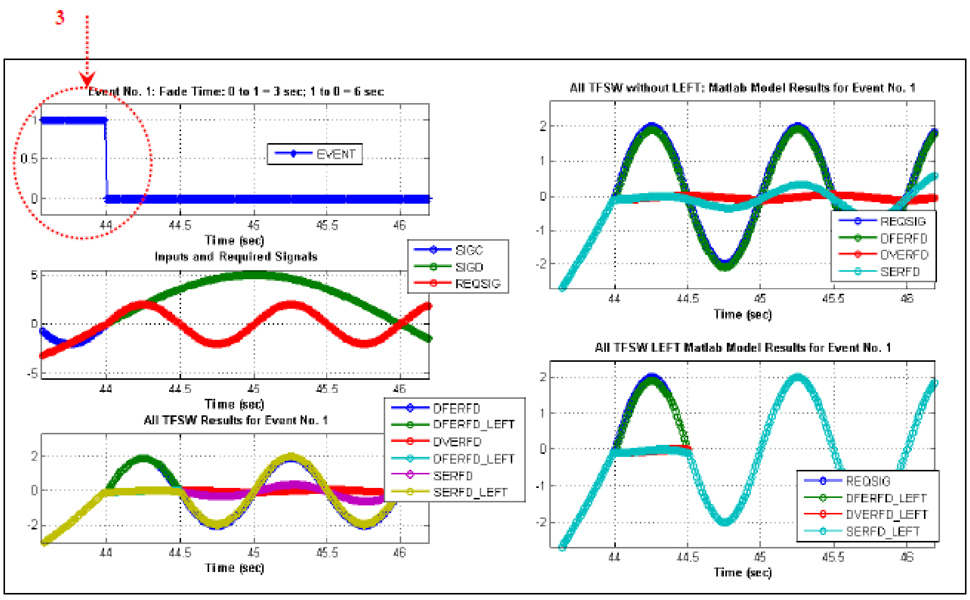


Figure 5: ALL TFSW Results, With and Without LEFT: Event Toggle Segment No. 3. Efficacy of LEFT Termination feature can be clearly seen in this Figure from the plots of with and without LEFT outputs (Selected Signals).