

28 June 1999

Windows 95/98 & Windows NT Compatibility

The SigGen Engine 32 (SGE32) is a System II software solution that provides, in conjunction with the S232 Device Driver & Controller, true 32-bit Windows performance. SGE32 allows the incorporation of SigGen32 files in the development of custom applications. In addition to offering better system compatibility, SGE32 is Windows 95/98 and Windows NT compatible. It incorporates a dynamic link library (DLL) as a standard interface tool. A DLL is a file that contains a library of procedures. The procedures of a DLL provide a collection of services through a well defined application programming interface (API). When applications use a DLL, they link a copy of the library at runtime and use those functions as though they were originally compiled with the executable.

There are a number of advantages to using DLLs.

- Multiple applications can share a single DLL simultaneously.
- Software upgrades become simpler. The applications that use DLLs do not need to be recompiled if the interface is standardized.
- Applications that use DLLs can be written in variety of different programming languages.

The following is the list of callable functions in SGE32.DLL that are described in *SigGen Engine User's Guide*.

```
void SetCallbackAddr(int funccode, LPVOID addr)
char *GetPrompt(int ctype, int n);
char *GetSrcstr(int ctype);
void FormFName(char sss[], SigType* Sig, CmpType* Cmp, int repn);
int CalcSteps(VarType *V);
void PullVars(SigType *Sig);
float GetVal(SigType* Sig, ParType p, int repn);
float ReadVarVal(SigType* Sig, ParType p);
void CalcSeg(SigType *TSig, int sn, int repn);
int GenGate(float *k, int n, float factor, float rf, float srate);
int CheckBoundry(SigType *Sig);
int CalcSig(SigType *Sig);
float SetAtten(SigType* Sig, float attadj);
int GetNewSeg(SigType *Sig);
void KillSeg(SigType *Sig, int sn);
void KillSig(SigType *Sig);
void DoNewSig(SigType *Sig);
void InitSig( SigType *Sig);
void ChangeSigNpts(SigType *Sig);
void InitSigStuff();
int SaveSig( SigType* Sig, const char *fname);
int LoadSig( SigType* Sig, const char *fname);
```

```

void SGfix4version(SigType* Sig);
void InitVars(SigType* Sig);
void PromptVars1(SigType* Sig);
void PromptVars2(SigType* Sig);
void LoadCSVFile(VarType* V);
int LoadSched( VarType* VT);
int ReadResponse(char *fname, float srate, int id1, int id2);
int LoadNorm(SigType* Sig);
int SetMasker(SigType* Sig, int mn, float addadj, int sumode);
void MaskerOff(SigType* Sig, int mn);
int SetTG6(SigType* Sig, int sn, float oodur, int reps, int extrig,
float os3, float dur3, float os6, float dur6, int microsec);

```

SGE32 Callback Support

SGE 32 requires that you develop some basic functions to accept user input, handle errors, and control dynamic variables. SGE32 expects these callback functions to detail how the application will interact with the user. In order to support these external callbacks, a new function, *SetCallbackAddr(int funccode, LPVOID addr)*, passes a callback function address to SGE32.

SetCallbackAddr(funccode, addr)

Prototype	void SetCallbackAddr(int funccode, LPVOID addr);														
Operation	Passes the address of the callback functions to the SGE32 dynamic link library (DLL). This provides the DLL with the location of a callable procedure.														
Arguments	<table> <tr> <td><i>funccode</i></td> <td>selects the callback function</td> </tr> <tr> <td><i>addr</i></td> <td>address of the function</td> </tr> <tr> <td>F_ERRHAND</td> <td>0</td> </tr> <tr> <td>F_ADKFORVAL</td> <td>1</td> </tr> <tr> <td>F_ASKFORFILE</td> <td>2</td> </tr> <tr> <td>F_GETDYNVAR</td> <td>3</td> </tr> <tr> <td>F_INITDYNVAR</td> <td>4</td> </tr> </table>	<i>funccode</i>	selects the callback function	<i>addr</i>	address of the function	F_ERRHAND	0	F_ADKFORVAL	1	F_ASKFORFILE	2	F_GETDYNVAR	3	F_INITDYNVAR	4
<i>funccode</i>	selects the callback function														
<i>addr</i>	address of the function														
F_ERRHAND	0														
F_ADKFORVAL	1														
F_ASKFORFILE	2														
F_GETDYNVAR	3														
F_INITDYNVAR	4														
Example	The following code declares and defines a function ErrHand(). SetCallbackAddr() gives the location of the procedure to SGE32.DLL. SGE32 can now execute the ErrHand() procedure in the calling application.														

```

void ErrHand(char s1[], char s2[]);
void ErrHand(char s1[], char s2[])
{
    char s[1000];
    strcpy(s,s1);
    strcat(s, "\n\n");
    strcat(s,s2);
    AfxMessageBox(s);
}

SetCallbackAddr(F_ERRHAND, &ErrHand);

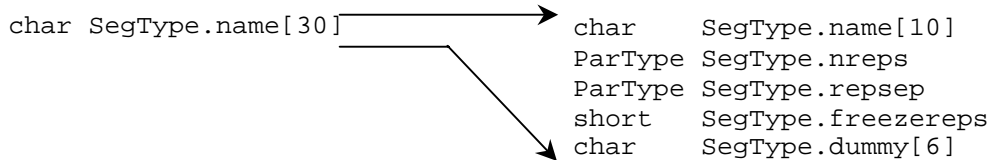
```

Support For New SigGen32 Features

The reader is referred to *SigGen32 New Features and Functions* for complete details.

Segment Repeating

The segment repeating variables are found within the SegType data structure. The memory for the new variables was previously part of the SegType.name. This approach preserves the size of the data structure and allows backward compatibility.

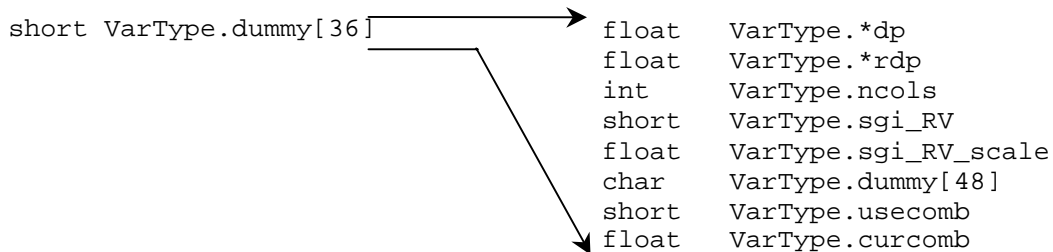


The new parameters directly control the behavior of the signal construction and are found in the segment design window of SigGen32. The functionality of each parameter is as follows:

SegType.name[10]	SegType variable name.
SegType.nrep	Number of segment repetitions in the overall signal.
SegType.repsep	Amount of time in milliseconds between the onset of each repetition.
SegType.freezereps	Does (zero) or does not (non-zero) recompute each repetition.
SegType.dummy[6]	For future use - DO NOT USE.

SGL Relative Indexing

The SGL relative indexing variables are found within the VarType data structure. The memory for the new variables was previously part of the VarType.dummy. This approach preserves the size of the data structure and allows backward compatibility.



VarType.*dp	Pointer to the beginning of the SGL variable matrix.
VarType.*rdp	Pointer to the beginning of a SGL variable matrix row.
VarType.ncols	Number of columns in the SGL variable matrix.
VarType.sgi_RV	SGL column indicator.
VarType.sgi_RV_scale	Scale factor for use in computing VarType.sgi_RV.
VarType.dummy[48]	For future use-DO NOT USE.
VarType.usecomb	Use the combination variable.
VarType.curcomb	Current value of the combination variable.

Extended Triggering Capabilities

The SetTG6() procedure has been extended to afford the programmer more control over hardware. Two new function argument have been included and are introduced below.

`int extrig` Sets the repetition mode to continuous (zero) or triggered (non-zero).
`int microsec` Sets the TG6 base rate to ms (zero) or μ s (non-zero).

SGE32 Implementation Changes

Allocating & De-allocating Buffers

SGE32 no longer allocates or de-allocates buffers in DAMA. This task is explicitly left under the direction of the application programmer. As a direct result, the previous structure variable SigType.id is no longer used.

`short SigType.id` \longrightarrow `short SigType.old_id`

The following procedures have been changed to reflect this new principle.

`KillSig(Sig)` Does not de-allocate the DAMA buffer for Sig.
`InitSig(Sig)` Does not allocate a DAMA buffer for Sig.
`ChangeSigNpts(Sig)` Does not de-allocate and re-allocate the DAMA buffer for Sig.

Additional Signal Calculations

SGE32 no longer calculates the signal and places it in the DAMA buffer indicated by the previously used SigType.id. This provides the programmer with the opportunity to perform additional operations on the computed signal before it is place in its DAMA buffer.

`CalcSig(Sig)` Leaves the computed signal on top of the stack.

