SGOS Source Code Structure & Insroduction

# SGOS 版本現況:

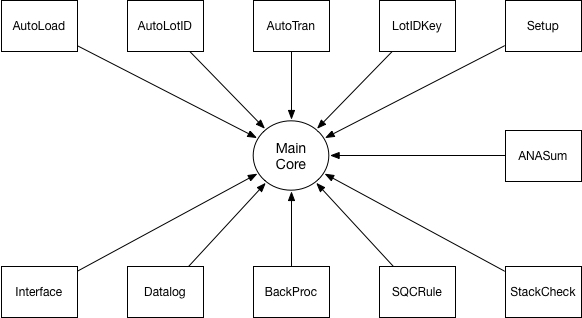
目前SGOS為對應各種(廠)測試流程而分出各種版本如下:

1. SGOS STDF V1.17.0 (中興廠, FT用)
2. SGOS Default V1.17.1 (北興廠, CP用)
3. SGOS MS (Multi Station用)
4. SGOS Testhead (for SOC device)

STDF, Default, MS等為SGOS在Mercurial Repository中的分支名稱。藉由Mercurial的分支功能，可以將各種版本歸在同一Repository下做管理，可避免程式碼在debug/verificatoin時混淆的風險。此三版統一可稱為Standard SGOS

SGOS Testhead 則為另一Repository管理，此版本專供有testhead的SG9000測試SOC類Device用，目前多半測MTK的產品。視為Testhead SGOS。

# SGOS Module 架構



SGOS程式架構將介面操作流程與測試程式呼叫保持在MainCore中， 而其他可分類的功能則以Module的概念製作為DLL檔案並統一放置在”\Interface”目錄下給Main Core呼叫執行。各Module之間以及Module與Main Core之間資料的傳遞則以結構 MyModuleVar 將所有需要的變數包成struct來做讀取。

各Module功能分類如下:

1. AutoLoad: Barcode file，Setup file讀取與下載測試程式
2. AutoLotID: 依照客戶的檔名規則產生測試檔案檔名， 為執行AutoLoad流程時所呼叫。
3. AutoTran: 檢查GS 結果並計算Transform values且存入檔案
4. LotIDKey: 使用Engineer頁面時按照客戶檔名規則產生檔名.
5. Setup: 讀取SPJT檔案所儲存的設定並apply至程式中
6. Interface: 負責與Handler/Prober的溝通，內含TTL與GPIB兩種溝通方式
7. Datalog: 將測試資料寫入STDF/XLS檔與產生LSR。
8. BackProc: 執行Part Average Testing limits計算與apply PAT limits. 舉凡每一DUT測試完畢後的資料相關額外處理皆可寫於此處。
9. SQCRule: 每一批測試完畢後檢查的Special yield control， 原本為給客戶(ANA)的客製化Module，未來可以考慮移除。
10. StackCheck: 以軟體方式的疊料檢查
11. ANASum: 為客戶(ANA)客製化的產生客製化format的Datalog，未來可考慮移除

將可分類的功能獨立為DLL Module，本是為了方便Maintain功能與更新。但因為FT, CP生產流程的不同，各Module與Main Core都會有些許的不同，若將FT, CP合併在一起的情況下，加上DLL Module架構下變數並無封裝 (物件也無法Re-use)，會有被錯誤修改的風險，犯錯的風險很大。因此目前將FT與CP測試分為兩版本管理。但因曾經嘗試合併版本，所以在source code可以中看到有部份兩種混雜在一起的地方。除了會慢慢清理分開這些合併部分以外，未來要合併必須要先經過物件概念(Object Orientation)的架構分析設計，才有機會合併且降低犯錯風險。或只至少要將程式邏輯與視窗操作介面邏輯分開，減少版本強行合併以及Maintain犯錯的風險。

# SOGS Module 結構變數

將SGOS各功能Module化獨立出去後，由於採用DLL (Dynamic Link Library)的方式，其變數傳遞不像物件方便及安全。但由於需求，因此將所有用到的變數以結構 (struct) 方式包裝，並以指標傳遞至各module。是故，所有 module共享一個 module variable。此 module 除了包含各種結構變數，連物件也包含在裡面以供各module傳遞使用。

Module變數實體宣告為”MyModuleVar”，而MyModuleVar包含的物件與結構變數內容則可在 “ClassSet.h” 此一header檔中看到。其中比較重要並且在各module傳遞的有以下資料結構物件(以下所列為已宣告出實體的物件名稱)：

1. testspec : 儲存測試程式中Table檔的test items資訊，以vector建立的class。
2. testlimit : 儲存測試程式中Limit檔的 testl limits資訊，以vector建立的class。
3. Config : 儲存測試程式與SGOS有關周邊設定的資訊
4. DUTInfo : 儲存測試程式SPJT檔中
5. TestVar : 與測試程式共享資訊的struct，測試資料的回傳也是透過此
6. BinCount : 紀錄Bin Summary的資料結構
7. BinLimit : 將limit file中記錄hard/software bin的相關資訊經過排序整理於此
8. SystemConifg : 記錄機台相關設定資訊，大多為SG9000II.ini內的設定
9. DatalogVar : DatalogModule 所需要從OS Main Core得到的資訊放於此struct
10. BackendProcVar : BackendProc Module所需要從OS Main Core得到的資訊放於此
11. DAQVar : Interface Module 所需要從OS Main Core得到的資訊放於此

struct TMyModuleVariable {

//----------------------------------OS Struct-----------------------------------

struct Count BinCount;

struct BinMapping BinLimit;

struct TestVariable TestVar;

struct OldTestVariable OldTestVar;

struct ConfigFlag Config;

struct BarCode Bar;

struct tagDUTInfo DUTInfo;

struct tagCheckStack CheckStack;

struct tagTimerStruct TimerStruct;

struct tagSystemConf SystemConfig;

struct tagDAQVar DAQVar;

struct tagDLLPath DLLPath;

//----------------------------------Module Struct-------------------------------

struct SetupVariable SetupVar;

struct LotIDKeyVariable LotIDKeyVar;

struct MonitorVariable MonitorVar;

struct SQCRuleVariable SQCRuleVar;

struct AutoLoadVariable AutoLoadVar;

struct DatalogVariable DatalogVar;

//------------------------------------------------------------------------------

struct vector<SPEC> testspec;

struct vector<Limit> testlimit;

struct vector<TestValueAnalyze> GoodParts;

struct BackupProcVariable BackupProcVar;

//------------------------------------------------------------------------------

struct SocketCtrlVariable SocketVar;

};

此MyModuleVar傳遞至Module的方式為透過指標，幾乎每一個Module都有個Initialize的呼叫函示，此時將指標傳遞給module例如: InitData(&MyModuleVar)，接著在module端將此指標儲存到module內的global指標如: MyModuleVar = tempModuleVar; 其中左邊MyModuleVar是module內的指標變數，右邊tempModuleVar是OS Main傳遞過來紀錄MyModuleVar的位址。如此在module內即可用MyModuleVar->的方式來存取struct內部的資料。

# Module 與DLL

SGOS中的Module皆以DLL (Dynamic Link Library)實作，並以動態載入的方式使用，在SGOS中使用方式如下:

1. 先宣告DLL中開放出的API函式型態 (以SQC Module為例)以及DLL指標 (HINSTANCE):

void (\_\_stdcall \*SQCRule)(TMyModuleVariable \*Tmp);

HINSTANCE SQCRuleDLL;

1. 接著載入DLL File 連結DLL中的函式指標:

SQCRulDLLPath = MyModuleVar.DUTInfo.OSPath +"\\include\\SQCRule.dll";

SQCRuleDLL = LoadLibrary(SQCRulDLLPath.c\_str());

(FARPROC&)SQCRule = GetProcAddress(SQCRuleDLL, "SQCRule");

便可使用SQCRule此一函式呼叫DLL中的動作

1. 使用完畢必須做解除動作:

FreeLibrary(SQCRuleDLL);

關於DLL的相關知識詳細介紹可參考:

<http://sun.cis.scu.edu.tw/~nms9115/articles/delphi/PluginDLL/PluginDLL.htm>

# Module 架構

某些Module因為其動作參與到整個測試流程，如Datalog，Interface，BackEndProc。故這類Module都會固定有 LotBegin, DutBegin, DutEnd, LotEnd四個類似函式。並將程式動作依照發生時間拆到這四個函式當中。可方便Maintain與編寫。

# IDE Tool for SGOS Development

SGOS開發的使用的工具平台一開始為Borland公司所開發的C++ Builder。後來Borland被Embarcadero收購後C++ Builder更名為 C++ Builder XE Series。兩者並不相容，故目前Maintain多半仍在Borland C++ Builder上，但此一開發平台已不在被更新支援。目前SGOS STDF (中興)有一轉移至XE2上的版本，但僅止於Offline環境驗證，還需經過線上驗證方能正式Release。此版目前當作萬一需要64-bits環境的SGOS時的備案，此版本可透過XE10做64-bits compile產生64-bits版本。未來若啟動物件化計畫，可考慮順便將平台移至XE Series，在物件化設計架構下，程式歸程式，介面歸介面，可以更容易移轉開發平台，降低移轉難度。

# SGOS 操作流程與執行流程

AutoLoad

Production

Engineering

Engineering

Setup

# \*Hardware Bin與Software Bin

在IC測試中，測試結果都會將IC做分類，分類稱作為Bin/binning。分Bin的設定一般由客戶指定，又分為Hardware Bin與Software Bin。

Hardware Bin: 配合機器如Handler/Prober上的實體分類，如Handler上裝有盛放分類IC的筒子，因handler可裝設的筒子數量有限，一般Hardware Bin會有數目限制。

Software Bin: 軟體上設定某一test item fail時便將IC分類為某個bin，此種bin值不需搭配機器限制，是故幾乎無上限，可任意指定方便分辨IC fail是在哪個test item。

# Table file & Limit file