台北大學資訊工程學系專題報告

Fault Scenario Generator with EN50159 for FlexRay

專題組員:涂仲蔚、黃培哲

指導老師:陳永源老師

專題編號: PRJ-NTPUCSIE-100-012
執行期間:100 年 2 月 至 101 年 1 月

1. 摘要

由於現今汽車電子的普及化，要將汽車電子運用在實際生活，其安全設計將會是一個大問題。一般環境下，可能會出現電磁波、雜訊等干擾，為了預防這些干擾，FlexRay 提供一個平台供我們設計錯誤去模擬錯誤環境。

在 FlexRay 的平台下，必須要手動去輸入 XML 製造錯誤環境，但是，若要大量產生 XML，此舉將會非常耗費時間、人力。為了解決此問題，實做一個 XML 自動產生器是我們這個專題的主要目的。

藉由 Visual Basic 語言，我們實做出 FlexRay 提供錯誤環境中的，Burst Error、Delay Model、Deletion Model 及混合型的錯誤環境。

2. 簡介

由於汽車電子悠關於駕駛人及乘客的生命安全，其可靠度必須經得起考驗，所以藉由 FlexRay 平台來模擬錯誤環境，驗證其強韌度是必需的。

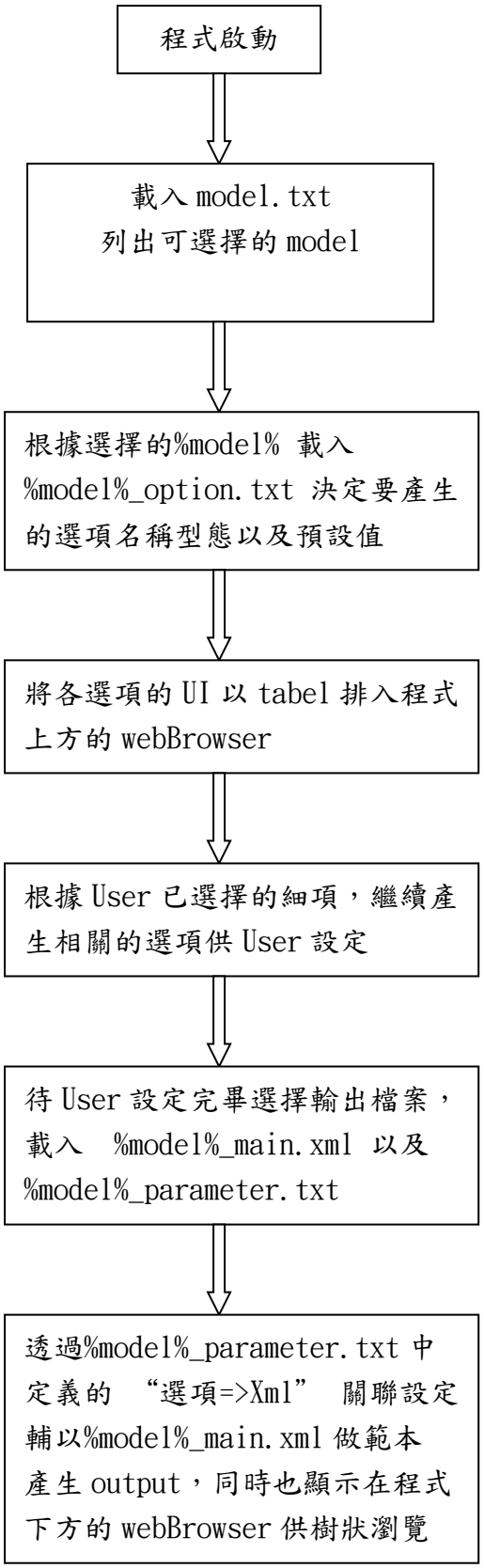
FlexRay 需要使用者自行輸入 XML 語法，一旦實驗需要多組 XML 時，使用者必須一組一組輸入，且輸入數據過於主觀，無法趨近於實際自然情況。所以，實做一個 XML 自動產生器去輔助實驗是目前當務之急。

利用 Visual Basic 配合 HTML 語法，我們將 FlexRay 的 Fault Model 定義在外部的 txt，由產生器解析後產生介面。程式讀取 model 設定檔後，自動產生 UI 的 html。定義 models 時，不需要撰寫 html 就可以設計 UI。而 UI 選項之間有樹狀關係，設計上更加靈活，可以根據不同情境對參數有不同的上下限或是預設值。

3. 專題進行方式

由於有許多的 fault model 必須實做，如果直接將產生器要產生的選項，以及輸出的轉換寫在程式碼中，日後需要修改選項或是設計新的 fault model 時會相當費時費力。

因此我們設計出了以下的架構來解決這個問題：



各 Model 設定請參閱 modelSetting 資料夾

關於ＵＩ定義：

Option.txt 中定義選項名稱,類型,預設值以及上下界。

Syntax

Option name////Default////Type////minimum////maximum

程式將指令以////分割為陣列作為參數傳入，根據參數產生對應的 html。

範例：

在此以 delay_option.txt 為例

較基本的如以下兩個範例

(1)Scenario name////Delay////TEXT////18

產生一個叫 Scenario name 預設值 Delay 長度為 18 的 TEXT

(2)wait_for_static_slot////1////number////1////4

產生名為 wait_for_static_slot，預設值１，且可設定 random 1~4 的數字選項

(3)delay_stream_unit////select_model////ns(01)////us(02)

產生下拉選單 delay_stream_unit 可選擇 ns 或者 us。

範例(3)用來說明，程式如何定義 UI 之間的樹狀關係：

若選 ns 則 程式中的 mode 變數改為 01 使得 option.txt 中開頭是(@01)的指令才被接受，若選擇 us 則是 (@02)。

因此若選擇 ns ,根據 (@01)delay_stream_value////260////number////260////6000

產生預設值 260 範圍限定 260~6000 的 value 選項。

若選擇 us 則產生範圍限定 1~6。

這項設計在 Corruption 中被大量使

用，由於 Corruption 分為
Noise_burst
Single_bit/single_frame
Single_bit/multiple_frame
Multiple_bit/single_frame
Multiple_bit/multiple_frame
五大類
其中 multiple_bit_error 的發生時間，又分成 User 自行設定的 manual 模式以及 Random 模式，因此 Corruption_option.txt 中用到許多範例(3)的指令來達到選項的分支。

特殊指令如
(@-loadModel-)////delay_main.xml
則是指定程式讀取 delay_main.xml 做為輸出範本，同樣支援@分支指令使得不同分支可以讀取不同範本。
如 Corruption 中 noise_bust 和 bit_error 就是讀取不同的 xml 做範本。

關於輸出定義：
同樣以 Delay Model 做例子
Delay_Parameter.txt 中定義 UI 選項，要對應到 XML 輸出檔的哪個位置。
而 Delay_main.html 則是一份 Delay_Model 的灌錯 XML 範例，並將各個 xml 節點加上唯一性的 ID 用來定位。
Syntax:
'id //// attribute name //// value
(1)d_s////unit////delay_stream_unit
將選項 delay_stream_unit 中的值填入範本中 id 為 d_s 的節點的 unit 參數
<delay_stream id="d_s" unit="us" value="260" />

(2)
d_s////value////number_of_continous_cycle////makeScenario
makeScenario 指令：
要求程式產生多個副程式，副程式的數量根據
number_of_continous_cycle 的值做設定
並指定產生的副程式中 d_s 的 value 值為可變動，再根據 parameter.txt：
d_s////value////delay_stream_value
得知 d_s 的 value 值為 delay_stream_value。

當 user 在 delay_stream_value 選項設定為 260 則所有副程式中的值都是 260，當設定 random 則所有副程式中的值都會 random 一個不重複的值。
其餘沒有在 makeScenario 中指定的參數，不論是 random 或是直接給值，都會照抄第一個副程式的值，例如第一個副程式 random 設定 duration 的 value，且隨機產生出 70 的值，接下來的其他副程式皆會直接使用 70 做設定。最後，將輸出結果顯示在程式中，以及輸出 xml 檔案供灌錯程式使用。

透過以上的架構結合 WebBrowser 並以 Html 做排版，我們可以定義出各種常用的選項ＵＩ。同時也支援 javascript 來設計 UI 被點選後的反應。最大的好處是，當內建的ＵＩ類型無法滿足需求時，option.txt 提供 LoadHtml 指令直接將外部 html 檔讀入做為 UI，最大限度的將ＵＩ設計及程式碼分離。

Multiple_bit 實做：
主要在 bit_stream 的 random 產生，其餘皆套用 option-parameter 的模組。

(1)Burst:

當程式收到 paremeter 指令,需輸出 bit_stream 時,若判斷為 Burst 模式

1. 程式取得 user 設定的 length n
2. 宣告長度為 n 的陣列 k
3. 將第 1 個及最後一個字元設定為 T
4. 利用迴圈將 2~n-1 個字元,以 50% 的機率隨機設定為 T 或 X
5. 將字串輸出到 bit_stream

(2)multipleBit

當程式收到 paremeter 指令,需輸出 bit_stream 時,若判斷為 multiple 模式

1. 取得 user 設定的 length n
2. 判斷為連續 or 非連續

連續:

直接輸出長度 n 的 bit_stream,內容皆為 T。

非連續:

產生兩段不連續的 error,意即兩段 error 中間至少會有一個 bit 不出錯,程式產生 2 段 error 如圖紅色部分,以三個參數 K, Kstart1, Kstart2 來描述兩段 bit-error

K:代表第一段 error 的長度,

Kstart1:表示第一段 error 的起始位置

Kstart2:表示第二段 error 發生的起始位置

而為了使每個 bit 產生的 error 發生機率相同以及顧全程式執行時間,在此列出我們實做的三種方法,其中 header 以及 payload 兩個環境參數,由使用者在一開始執行程式時設定,而 errorCount 是錯誤的 bit 數,必須由使用者自訂

1. 列舉全部

程式列出所有可能的 K, kStart1, kStart2 組合,再由組合中隨機抽出一種。以 errorCount=5、header = 5byte、payload = 34byte 為例,此例共有 189111 種組合,再從這組合數中隨機挑出一個。

2. 依序 Random

程式先 random 產生 1~n 的 K 值再 random 產生 kStart1、kStart2 值

3. 新方法

令 L 為 header+payload 之 bit 數,n 為 errorBit 數

a. 計算組合數

(a) 0 <= kStart1 <= L - (n-1)

(b) kStart1 + k + 1 <= kStart2 <= L - (n-k)

(c)令 F(x)=(當 K=x 時,(kStart1, kStart2)的組合數)由(a)、(b)中可以看出 F(0) = F(1) = F(2) ...= F(x)

(d) F(x) = 底為 (L-n),高為 (L-n),之三角形面積

b. 隨機選出一種組合

將每一種組合視為一顆寫上編號之球 , 共有 totalCount 顆,因此所有球可以依序排列成 n-1 個三角形,每個三角形為底 L-n,高 L-n。

現隨機取出一球，其為第A個三角形，第B層，第C顆，計算ABC之值。

c. 實作

(a) 程式隨機產生一值 R

0 <= R <= totalCount

(b) 計算A值

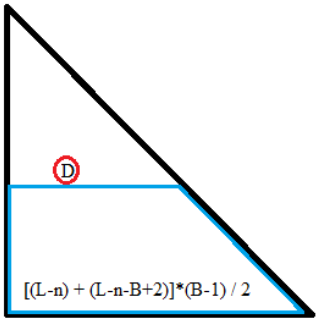
A = [R / F(x)]+1

(c) 計算BC值

若第A個三角形之第D顆球 = 第A個三角形之第B層第C顆球

則 D = R mod F(x) + 1

如下圖所示



第(1~B-1)層共有 [(L-n) + (L-n-B+2)]*(B-1) / 2 顆球

故 [(L-n) + (L-n-B+2)]*(B-1) / 2< D，L、n、D為已知，求B

(1) B^2 - (2*(L-n)+3)B + (2*(L-n)+2+2*D) > 0

(2) 1<=B<=L-n，B為整數

利用一元二次公式解(1)的式子求出兩個解，其中符合(2)式子者為B值。已知B值，即可求出C值

C = D - (2 * (L-n) - B + 2) * (B - 1) / 2

d. 計算k, kStart1, kStart2

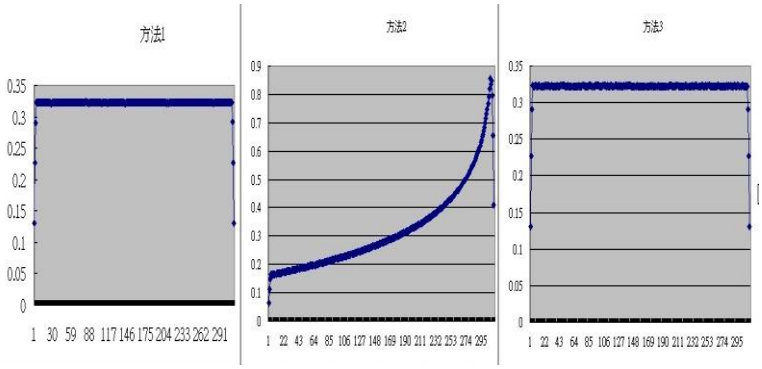
K = A

kStart1 = B - 1

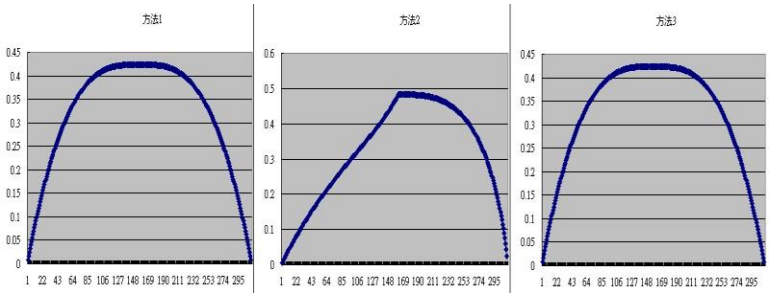
kStart2 = K + kStart + C

而以下分別測試三種方法在errorCount=5bit以及errorCount=150bit時的比較(此時環境參數header = 5byte、payload = 34byte)

(1)在errorCount=5bit的情況下，模擬一千萬次，第1個bit~第312個bit發生error的機率如下圖，x軸為第n個bit，y軸為error發生機率，單位為(%)



(2) 在errorCount=150bit的情況下，模擬一千萬次，第1個bit~第312個bit發生error的機率如下圖，x軸為第n個bit，y軸為error發生機率，單位為(%)



由上列的圖來探討每個bit發生錯誤的分布狀況，使用方法1(列舉全部)，不論errorCount為多少，錯誤發生機率呈現對稱分佈，但是左右兩端的bit發生error的機率較低，此現象，是因為必須產生兩段bit-error不重疊且不連續的限制造成的特性。方法2(依序

random)，結果會隨著 errorCount 改變，而影響 bit-error 分布的狀況。而方法 3(新方法)，不論 errorCount 為多少，亦如同列舉全部的方式呈現對稱分佈。

而下表為測試時所花費的時間，單位(秒)

組合數(百萬)	0.4	2.2	4.18	10	20	40	60	80	100	500
方法 1	2	4	5	8	17	N/A	N/A	N/A	N/A	N/A
方法 2	2	2	2	2	2	2	2	2	2	2
方法 3	4	5	4	5	5	4	4	5	4	4

由此表來探討各方法的花費時間
方法 1(列舉全部)，隨著組合數的上升，時間的花費一開始是小幅度的增加，當組合數越來越高，其時間成長呈現指數成長，此方法最後已不敷使用，測試時直接造成記憶體不足。而方法 2(依序 random)，其時間花費並不會因為其組合數增加而增加。方法 3，其時間花費，也並不會因為其組合數增加而上升。

為了驗證其花費時間的正確性，以下分析其時間複雜度

首先定義相關參數如下
L = payload+header 長度
N = 需產生的 bitError 長度
TC = 對應 L,N 的參數組合總數 =[(L-n) * (L-n) / 2] * (n-1)

(1)列舉全部
迴圈列舉 k = 1~n{
　迴圈列舉 kStart = 1~n{
　　迴圈列舉 kStart2 = 1~n{
　　紀錄 k,kStart,kStart2 值之組合
　　總組合數 + 1
　　}
　}
}
產生的 R 值介於 1~總組合數，取出第 R 個組合之 k, kStart, kStart2 值

已知參數組合共有 [(L-n)*(L-n)/2] *(n-1) 種，因此程式使用迴圈列舉出所有組合，需執行 TC 次迴圈，其時間複雜度為 O([(L-n) * (L-n) / 2] * (n-1)) =O(L^2+n^3)

迴圈中的兩個步驟，時間複雜度為 O(1)
產生 R 值及取出第 R 個組合，時間複雜度為 O(1)，故此方法為 O(n^3)的演算法
(2)依序 Random
具體執行的程式步驟為
a. 隨機產生 k 值 介於 1 ~ n
b. 計算 kStart 之上限 L-n-1
c. 隨機產生 kStart 值 介於 0 ~ L-n-1
d. 計算 kStart2 之下限
　[kStart+k+1]
e. 計算 kStart2 之上限 [L- (n-k)]
f. 隨機產生 kStart2 值介於(4)(5)之值

由於步驟 a, c, f 隨機產生一數所需的時間複雜度為 O(1)，而步驟 b, d, e 之計算時間複雜度也是 O(1)，因此方法二總共需六個步驟 其時間複雜度為 O(6) =O(1)

(3)新方法
a. 計算單 1 個 k 值所對應的 kStart,kStart2 組合數 F(x)=[(L-n)

*(L-n) / 2]

b. 計算總組合數 TC = F(x) * (n-1)

c. 隨機產生一數R 介於 1 ~ TC

d. 計算A值= ⌊R / F(x) ⌋+1

e. 利用公式解求出符合以下兩式之B值

(1) B^2 - (2*(L-n)+3)B + (2*(L-n)+2+2*D) > 0

(2) 1<=B<=L-n，B為整數

f. 計算 D = totalCount mod F(x) +

g. 計算 C = D - (2 * (L-n) - B + 2) * (B - 1) / 2

h. 計算第 R 個組合之K值 = A

i. kStart = B -1

j. kStart2 = K+kStart+C

以上10個步驟, 時間複雜度皆為O(1)，因此方法3整體之時間複雜度亦為O(1)，即不論L、N兩項參數數字多大，所需的處理時間皆不受影響

下表整理三個方法的時間複雜度

	速度	ErrorBit分佈機率
方法1	過慢，Big0=n^3	平均且對稱
方法2	快 , Big0= 1	隨errorCount改變
方法3	快，Big0= 1	平均且對稱

分析後發現，方法1雖然達到幾乎每個bit發生error的機會相等，但是相對的，再多種組合數的情況下，因為需要完全展開，花費的時間相對較高，在高組合數下，甚至是太過於吃資源以至於無法運算。而方法2，其每個bit發生的機率要在多errorCount下才能接近方法1近乎平等的機率分布，但是其時間複雜度為常數，即便是在高組合數下，依然是可以在很快的時間內算出來。而我們綜合以上兩方法的優點，做出了方法 3，並以實驗及分析演算法去佐證它，不僅顧全了機率分布，時間複雜度也維持在常數。

Delay model 實做 :

重點在多scenario的產生，介面及輸出皆套用option-parameter的模組即可。當程式從parameter.txt接到makeScenario指令。

d_s ////value////number of continous cycle ////makeScenario

取得user設定的number of continous cycle值 N,以及scenario name S

1. 複製xml中第一個scenario，產生N-1個scenario，命名S2 S3 S4...SN。
2. 對新產生的N-1個scnario，根據user對delay_stream_value的設定做不同應對。

Delay_stream_value 非random :

不須修改，N個scenario使用相同值

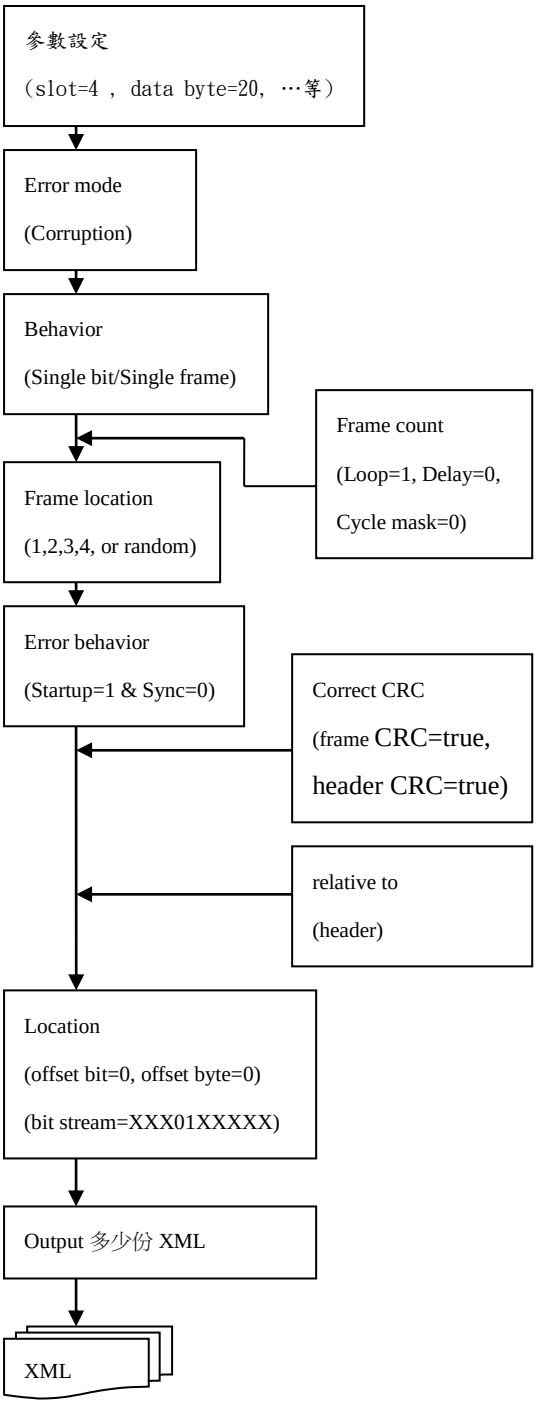
delay_stream_value 為random :

以user設定的範圍隨機產生值，填入新產生的scenario,並記錄此隨機值，若和其他scenario重複,且N <= 隨機的可能數則重新產生。若user設定10個scenario,但value設定1~6隨機，則程式不會避開重複的value值。

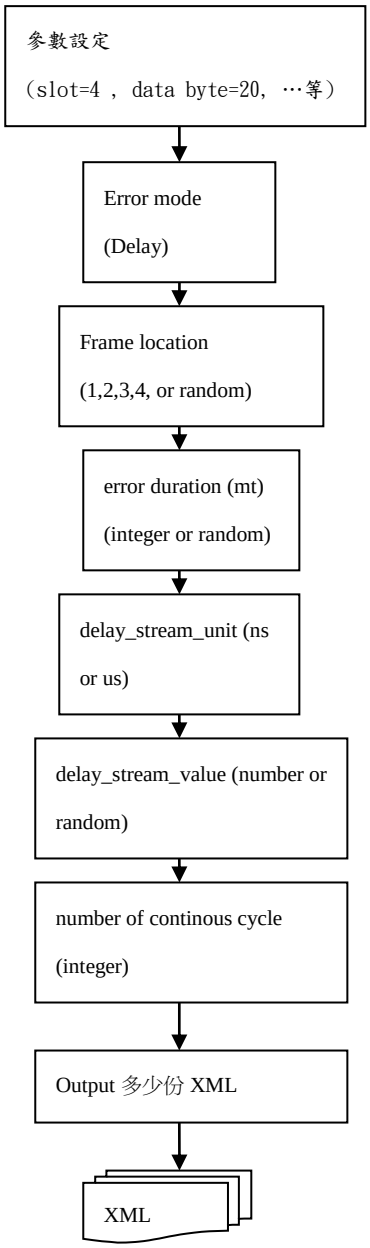
設定完所有scenario後，產生main scenario中的N-1條 run scenario指令 ，對應新產生的 N-1 個

scenario S2.S3.S4...SN。

Corruption Model 架構圖



Delay Model 架構圖



由上述的範例可以看出，各個 fault model 之間所需的選項以及輸出的對應相差非常大，也有可能會需要增減修改。這也是為什麼我們選擇大費周章的將程式設計成讀取 txt 中的自定義指

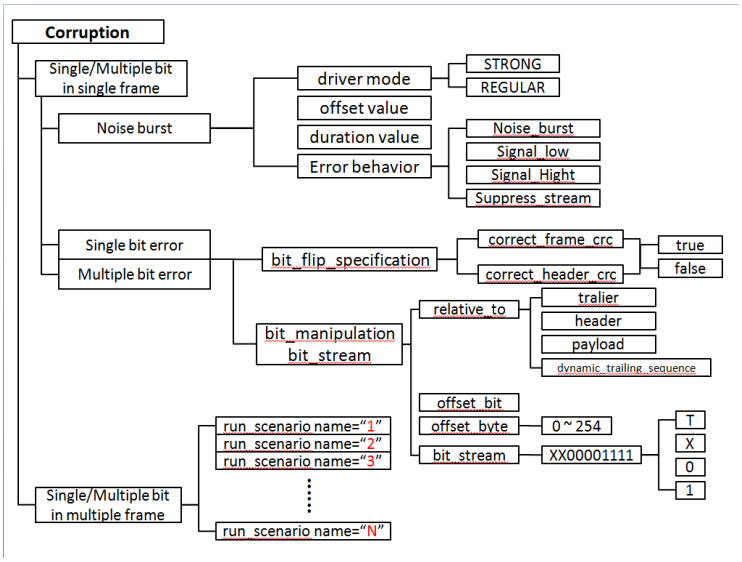
令，再轉成ＵＩ。

目的就是希望日後維護以及交接的時候，可以最小幅度的修改程式碼，減少維護程式所需的人力及時間。

4. 主要成果

以下展示各個 model 的範例

Corruption:



根據此流程圖實做

Noise_burst

single_bit/single_frame

Multiple_bits/single_frame Burst

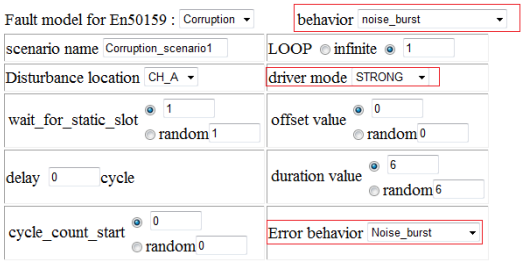
Multiple_bits/single_frame MultipleBits

Delay:

One-cycle

Multiple-cycle

1.Noise_Burst :

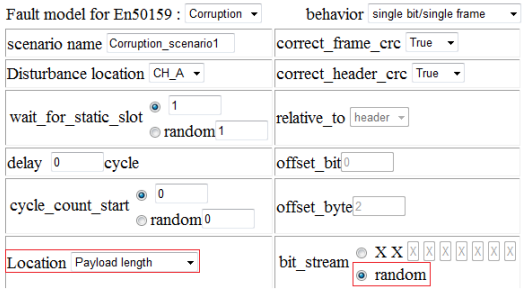



```
- <SCENARIO name="main">
    <FLEXRAY_SETUP transmission_speed="10" />
    <RUN_SCENARIO name="Corruption_scenario1" loop="1" />
  </SCENARIO>
- <SCENARIO name="Corruption_scenario1">
    <WAIT_FOR_STATIC_SLOT cycle_count_mask="1" slot="1" />
    <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
    <OFFSET value="0" unit="MT" />
  - <DISTURBANCE>
      <DRIVER mode="STRONG" />
    - <CHANNEL CH_name="CH_A">
        <DURATION value="6" unit="us" />
        <DELAY value="0" unit="ms" />
        <NOISE_BURST />
      </CHANNEL>
    </DISTURBANCE>
```

（圖 1）

根據(圖 1)設定的參數產生對應的 xml。

2.Single_bit/Single_Frame:



（圖 2）

根據（圖 2）由於 Location 選擇 PayLoad length，程式自動將 offset_bit/offset_byte 設定在對應的位置，bit_stream 選擇 random 輸出如(圖 3)。

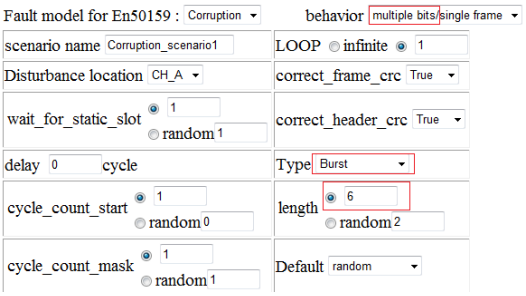
```
<SCENARIO name="Corruption_scenario1">
  <WAIT_FOR_STATIC_SLOT slot="1" />
  <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
- <DISTURBANCE>
  - <CHANNEL CH_name="CH_A">
    - <BIT_FLIP_SPECIFICATION correct_header_crc="True" correct_frame_crc="True">
        <BIT_MANIPULATION relative_to="header" offset_byte="2" offset_bit="0"
          bit_stream="XXXT" />
      </BIT_FLIP_SPECIFICATION>
    </CHANNEL>
  </DISTURBANCE>
  <TRIGGER_OUT name="TRIGGER_1" switch="OFF" />
</SCENARIO>
```

（圖 3）

程式隨機產生 bit_stream, 字串尾部連續的 X 被去除，crc 則是根據 input 設定。

3.Multiple_bits/single_frame:

Burst



（圖 4）

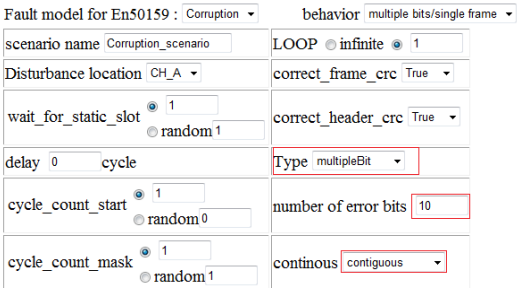
依據 multipleBit burst 的定義，產生總長度 6，頭尾為 T 且第 2~5bit 為隨機 T 或 X 的 bit_stream=>T _ _ _ _ T，如(圖 4)

Default 選擇 Random，代表由程式隨機決定錯誤發生的位置，結果如(圖 5)。

```
- <BIT_FLIP_SPECIFICATION correct_header_crc="True" correct_frame_crc="True">
    <BIT_MANIPULATION relative_to="payload" offset_byte="111" offset_bit="2"
      bit_stream="TTXXXT" />
  </BIT_FLIP_SPECIFICATION>
```

（圖 5）

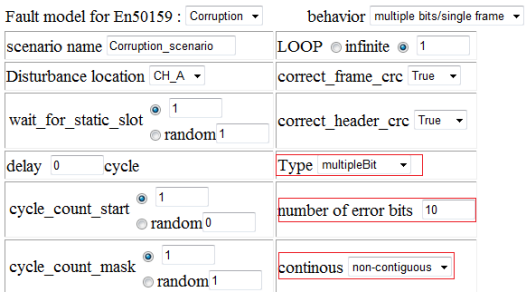
Multiple Bit：



（圖 6）

產生連續 10 個 bitError 如(圖 6)，位置隨機。

```
<DISTURBANCE>
- <CHANNEL CH_name="CH_A">
  - <BIT_FLIP_SPECIFICATION correct_header_crc="True" correct_frame_crc="True">
      <BIT_MANIPULATION relative_to="payload" offset_byte="63" offset_bit="0"
        bit_stream="TTTTTTTTTT" />
    </BIT_FLIP_SPECIFICATION>
  </CHANNEL>
</DISTURBANCE>
```



（圖 7）

產生 10 個非連續 bitError，可能是(1bit+9bit) or (2bit+8bit)...etc 如(圖 7)

```
<SCENARIO name="main">
  <FLEXRAY_SETUP transmission_speed="10" />
  <WAIT_FOR_POC_STATE state="POC:normal_active" />
  <ENTER_BUS_SEPARATION_MODE />
  <RUN_SCENARIO name="Corruption_scenario" loop="1" />
  <RUN_SCENARIO name="Corruption_scenario2" loop="1" />
</SCENARIO>
```

（圖 8）

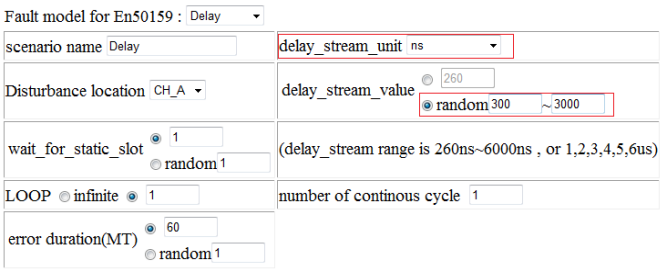
Two scenario for (kbit,n-kbit) 如(圖 8)

```
- <SCENARIO name="Corruption_scenario">
    <WAIT_FOR_STATIC_SLOT cycle_count_mask="1" slot="1" />
    <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
  - <DISTURBANCE>
    - <CHANNEL CH_name="CH_A">
      - <BIT_FLIP_SPECIFICATION correct_header_crc="True" correct_frame_crc="True">
          <BIT_MANIPULATION relative_to="payload" offset_byte="230" offset_bit="4"
            bit_stream="TTTTTT" />
        </BIT_FLIP_SPECIFICATION>
      </CHANNEL>
    </DISTURBANCE>
    <TRIGGER_OUT name="TRIGGER_1" switch="OFF" />
  </SCENARIO>
+ <SCENARIO name="Corruption_scenario2">
    <WAIT_FOR_STATIC_SLOT cycle_count_mask="1" slot="1" />
    <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
  - <DISTURBANCE>
    - <CHANNEL CH_name="CH_A">
      - <BIT_FLIP_SPECIFICATION correct_header_crc="True" correct_frame_crc="True">
          <BIT_MANIPULATION relative_to="payload" offset_byte="225" offset_bit="5"
            bit_stream="TTTT" />
```

（圖 9）

範例為 6bit+4bit，兩 error 區段不重疊且不連續如(圖 9)。

Delay model:



（圖 10）

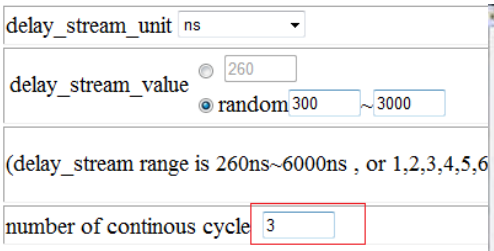
設定延遲的時間及單位如(圖 10)

因為 Cycle 數為 1 只產生一個 scenario 如(圖 11)

```
<DISTURBANCE>
- <CHANNEL CH_name="CH_A">
    <DURATION value="60" unit="MT" />
    <DELAY_STREAM value="430" unit="ns" />
  </CHANNEL>
</DISTURBANCE>
```

（圖 11）

調整 cycle 數為 3 如(圖 12)



（圖 12）

產生 3 個 scenario，隨機且不重複產生 3 delay_stream_value，如(圖 13)

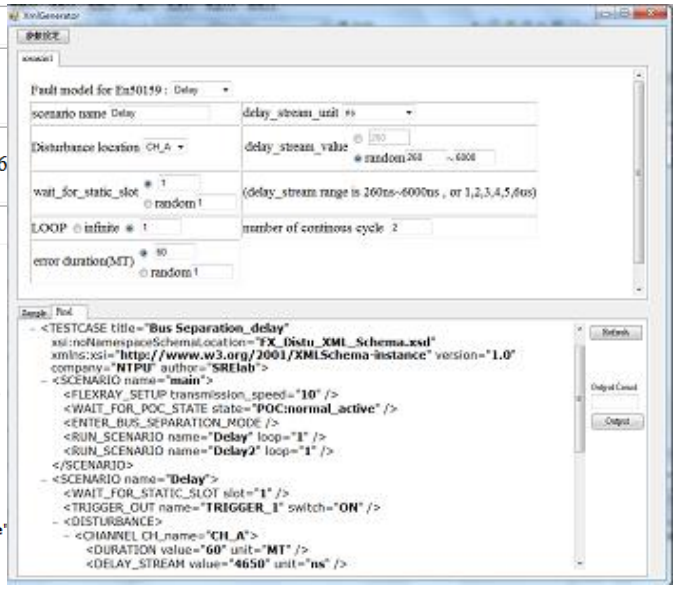
```
<SCENARIO name="main">
  <FLEXRAY_SETUP transmission_speed="10" />
  <WAIT_FOR_POC_STATE state="POC:normal_active"
  <ENTER_BUS_SEPARATION_MODE />
  <RUN_SCENARIO name="Delay" loop="1" />
  <RUN_SCENARIO name="Delay2" loop="1" />
  <RUN_SCENARIO name="Delay3" loop="1" />
</SCENARIO>

<SCENARIO name="Delay">
  <WAIT_FOR_STATIC_SLOT slot="1" />
  <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
- <DISTURBANCE>
  - <CHANNEL CH_name="CH_A">
      <DURATION value="60" unit="MT" />
      <DELAY_STREAM value="2260" unit="ns" />
    </CHANNEL>
  </DISTURBANCE>
  <TRIGGER_OUT name="TRIGGER_1" switch="OFF" />
</SCENARIO>
```

```
<SCENARIO name="Delay2">
  <WAIT_FOR_STATIC_SLOT slot="1" />
  <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
- <DISTURBANCE>
  - <CHANNEL CH_name="CH_A">
      <DURATION value="60" unit="MT" />
      <DELAY_STREAM value="2080" unit="ns" />
    </CHANNEL>
  </DISTURBANCE>
  <TRIGGER_OUT name="TRIGGER_1" switch="OFF" />
</SCENARIO>
<SCENARIO name="Delay3">
  <WAIT_FOR_STATIC_SLOT slot="1" />
  <TRIGGER_OUT name="TRIGGER_1" switch="ON" />
- <DISTURBANCE>
  - <CHANNEL CH_name="CH_A">
      <DURATION value="60" unit="MT" />
      <DELAY_STREAM value="1270" unit="ns" />
```

（圖 13）

程式完整畫面



目前程式完成了主要的架構，接下來就是設計各個 model 的設定檔，並設計”project”功能，以便 user 將當前的設定存為一個 project 檔，程式讀取該檔案即可載入先前 user 設定的值。

5. 評估與展望

目前整個程式的功能已趨於完整，除了原先預定的單一 fault model 之外，也新加入了在每個 scenario 中灌不同錯誤的情境。

未來可以結合資料庫，收集曾經測試過的錯誤環境，配合實驗結果建

檔，建立Fault library。

6. 結語

希望這個Generator將原先需要人力輸入指令的步驟自動化，可以減少開發人員灌錯測試時浪費不必要的時間及人力，程式隨機產生出的XML也使測試更貼近實際自然的狀況，同時考量到程式的執行時間，並導入了程式測試，確保程式的可靠度，可以讓使用者在不費時、不主觀及可信任的前提下，大量產生錯誤環境。

7. 銘謝

首先要感謝陳永源老師的用心指導，給了我們很多想法，以及思考的方法。此外，還要感謝學長們，費心思的跟我們討論程式的一些參數設定以及介面相關問題。因為有大家才可以讓我們的程式慢慢的更趨完整。

8. 參考文獻

2011.3.4Test_Case_for_EN50159.ppt

Disturbance_Node_user_man_v1[1].0.10.pdf

陳永源老師—Fault_generator.pdf