適用於多核心應用程式開發之圖形化工具

A Graphical Tool for Multicore Application Programming

莊景翔 Ching-Hsiang Chuang 陳秋伶 Chiu-Ling Chen 黃保瑞 Pao-Jui Huang
林泰吉 Tay-Jyi Lin 朱元華 Yuan-Hua Chu
處理器與應用組

摘要

模組化的軟體開發已被公認為簡化多核心應用程式發展最有效的方法之一，尤其是被應用在串流多媒體應用開發時。但編輯軟體模組以架構應用系統時仍嫌繁瑣費時，其中人為錯誤的發生機會亦不小。本篇論文提出了一個圖形化的串流應用程式開發工具，讓開發者能以直覺的方式開發多核心的串流應用程式，並於 TI DM6446 DVEVM 硬體平台上執行。本論文以 JPEG 編碼器為範例，將所提出之多核心軟體開發方式與傳統手動編輯的方式實作的串流應用程式進行效能上的比較。實驗數據顯示，採用本工具進行圖形化軟體開發流程，只會在程式執行期增加約1%的效能損失，這在實際應用上是可被忽略的。

1. 前言

愈來愈多的手持式裝置要求整合多媒體串流應用，這帶給了嵌入式系統額外的效能挑戰。而嵌入式系統本身具有功耗限制，如何在效能與功耗上取得平衡點是個嚴苛的考驗。多核心架構是一個針對這個問題的解決方式，它能夠在低功率消耗的情況下，提供足夠的計算能力[1]。然而多核心架構也帶來開發多核心應用程式的挑戰。

模塊化的軟體開發是大家公認能夠有效簡化開發多核心應用程式開發的方法。市面上不少商用軟體早已提供圖形化的工具協助模塊化的軟體開發流程，大幅簡化建立應用程式的複雜度，如Simulink [2]與LabView [3]。Simulink把許多演算法包裝成一個個的軟體元件，供軟體開發者在圖形化工具上組合。組合出的應用程式能夠於嵌入式系統上運作，不過開發者只能使用Simulink內訂的元件，無法自行新增嵌入式平台相關的元件，錯過了元件最佳化的機會。LabView雖然支援多核心的架構，不過只侷限於同質性多核心的x86 主機，不是專為嵌入式系統所設計。

本論文使用TI DaVinci [4]作為多核心的開發平台，並且使用TI Codec Engine [5], [6], [7] 作為遠端程序呼叫的軟體框架，來開發模塊化的串流應用程式。然而使用TI Codec Engine並不直覺，建置過程也相當繁瑣，直接使用會有入門門檻，造成開發者的困擾。傳統手動編輯串流應用程式的方法，也難免因為過程繁瑣費時，而造成人為錯誤。舉例來說，程式設計師在建立或調整緩衝區時，就必須依據不同變動重新調整程式碼，這對軟體最佳化的過程來說相當地不便利。一些常常發生的程式撰寫錯誤，如語法、語意上的錯誤也會在這個過程中出現，造成程式開發者

的困擾。除此之外，軟體開發者也缺乏一個有效率的方式為此類串流應用程式進行除錯。

有鑑於此，本論文提出 Multicore Development Suite(MDS)以解決這些問題。本論文的主要貢獻如下：

- 一個圖形化的工具協助快速的模塊化串流應用程式開發，它提供多核心軟體開發者一個視覺化的程式開發環境，並且能提供元件層級的除錯功能。
- 一個自動產生目標平台程式碼的工具。它能夠把 MDS 圖形化工具開發的多核心應用程式，轉換成能夠在 TI DM6446 DVEVM 平台上執行的應用程式。

本論文後面的章節規劃如下：在第二章中，我們會介紹 TI Codec Engine，簡單介紹如何使用 TI Codec Engine 進行模塊化應用程式的開發，並且指出其缺陷。在第三章中，我們提出 MDS 一個圖形化的模塊化應用程式開發工具，供開發多核心應用程式使用。在第四章中，我們做了兩個實驗來展示 MDS 的功能。第五章是本論文的結論。

2. 使用 TI Codec Engine 進行模塊化的軟體開發

A. TI Codec Engine 簡介

TI DM6446 DVEVM是TI所開發的高效能異質性多核心平台，平台上有一個負責控制程式流程的ARM，與一顆用來作運算的DSP。TI提供Codec Engine軟體框架簡化遠端程序呼叫的行為(RPC)，讓軟體開發者在程式中直接以函數呼叫的方式，在DSP上執行符合eXpressDSP [8], [9]規格的演算法。在本篇論文中，所有符合eXpressDSP規格的演算法，我們稱它作為codec。在TI Codec Engine的軟體框架下，程式開發者使用C語言撰寫應用程式，將codecs視為函式庫。最後使用XDC-Tool[10]協助建置整個應用程式。

使用 TI Codec Engine 的優勢是程式開發者在撰寫程式碼時，不需要管理 codec 的執行方式，例如決定該 codec 是在 ARM 端執行(近端執行)或是在 DSP 端執行(遠端執行)的動作。它把複雜的遠端程序呼叫行為抽象化，讓程式開發者不需修改原本的應用程式，就能決定 codecs 執行的方式。

圖一顯示TI Codec Engine在異質性多核心系統上的角色，對於程式開發者，它提供VISA [12]標準的應用程式開發介面(API)。只要符合VISA介面的codecs都可以透過TI Codec Engine以函式庫的方式呈現給程式開發者。在codecs的底層呼叫上，則由TI Codec Engine決定要由ARM直接在使用者空間(User Space)執行；或是透過作業系統核心空間(Kernel Space)下的驅動程式執行DSP上的codecs。DSP端有一個即時的作業系統，DSP/BIOS ™ [11]負責處理來自ARM的codec執行需求。

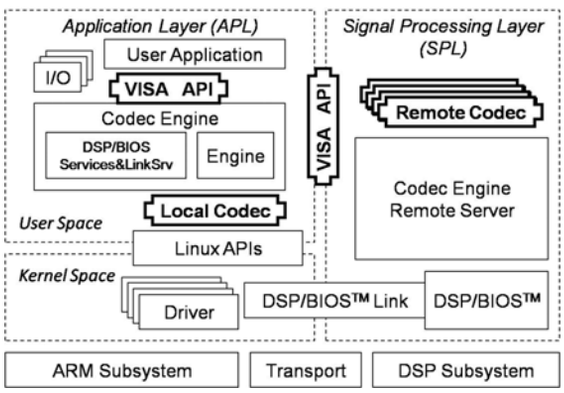
參考圖二，使用 TI Codec Engine 時必須釐清四個使者角色：

Role 1. Algorithm Creator – 建立符合 xDAIS 標準演算法的程式開發者。

Role 2. Server Integrator – 建立支援遠端 codecs 執行程式(稱為 Codec Server)的程式開發者。

Role 3. Engine Integrator – 設定 TI Codec Engine 應用程式建置組態的程式開發者。

Role 4. Application Author – 使用 TI Codec Engine APIs 建立、刪除、初始化 codecs，以及利用緩衝區將這些 codecs 組成應用程式的程式開發者。



圖一　TI Codec Engine 在異質性多核心系統上的角色

B. JPEG 編碼器實作

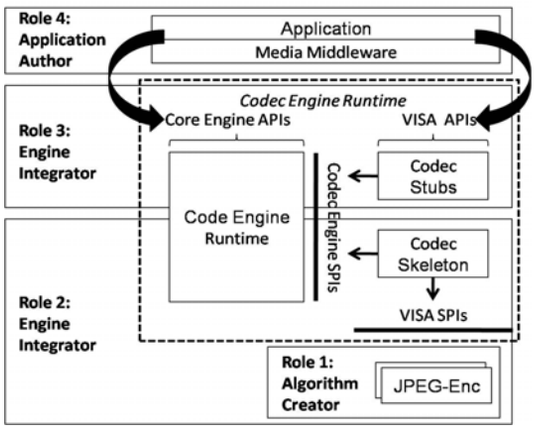
這裡我們採用 JPEG 編碼器作為使用 TI Codec Engine 建立模塊化串流應用程式的範例。請參考圖三，JPEG 編碼器由四個元件所組合而成：顏色域轉換元件(CST)、離散餘弦轉換元件(DCT)、量化元件(QUANT)、與赫夫曼編碼元件(HUFF)。

程式開發者必須先把幾個元件分別實作成四個 codecs。接著應用程式開發者撰寫另一支程式組合這些元件成為 JPEG 編碼器串流應用程式，部份的程式碼節錄在圖四中。這段程式碼只展示與串流應用程式相關的部份，並不包含使用 TI Codec Engine 過程中大量的例行動作與程式建置相關的設定檔。

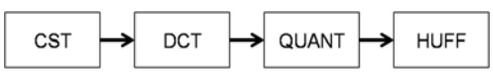
經過上面的步驟之後，程式開發者必須針對這四個 codecs 與一個串流應用程式建立對應的 Engine 組態與設定 Codec Server。Engine 組態用於串流應用程式建置的過程，其內容包含每一個 codec 的執行方式設定，因此在應用程式的建置過程中就限制了 codecs 的執行方式。相對應地，Codec Server 則要配合 Engine 組態，蒐集所有將會在 DSP 上執行的 codecs，組合成一支在 DSP 上的應用程式 – Codec Server。圖五是 Codec Server 組態設定的片段，裡面包含一些細微的 Server 參數設定例如：執行緒堆疊大小、執行緒優先順序。其中最重要的就是指定 Server 所要包含的 codecs 與其執行緒參數設定。圖六則是 Engine 組態的範例，內容描述使用 TI Codec Engine 程式開發者能在組合應用程式時使用的 codecs 與其執行方式 : 遠端(local : false)或是近端執行(local ： true)。除此之外，在 Engine 組態檔中，還必須指定其對應的 Codec Server 二進制檔(./app.x64P)。

建置與執行整個 TI Codec Engine 應用程式的程序如下：

1. 建置四個元件 {CST, DCT, QUANT, HUF }，產生 DSP 的 c6x 二進制檔與 ARM 的二進制檔。
2. 建置 Codec Server，論文中範例輸出檔為 “all.x64”。
3. 建置應用程式，論文中範例輸出檔為“app.out”。
4. 將 app.out 與 all.x64 複製到 DVEVM 板子上然後執行 app.out，即可執行此串流應用程式。



圖二　TI Codec Engine 中四個使用者角色



圖三　JPEG 編碼器的工作圖與資料流圖

```
// Read the picture
ReadPicture(&buf_0);
// Do JPEG-Encoding
CSTInArgs.inBufSize = FRM_SIZE;
CSTInArgs.outBufSize = FRM_SIZE;
CSTInArgs.inBufValidBytes = FRM_SIZE;
CST_process( cst_comp,  buf_0, buf_1,
   &CSTInArgs, &CSTOutArgs );
DCTInArgs.inBufSize = FRM_SIZE;
DCTInArgs.outBufSize = FRM_SIZE;
DCTInArgs.inBufValidBytes = FRM_SIZE;
DCT_process( dct_comp,  buf_1, buf_0,
   &DCTInArgs, &DCTOutArgs );
QUANTInArgs.inBufSize = FRM_SIZE;
QUANTInArgs.outBufSize = FRM_SIZE;
QUANTInArgs.inBufValidBytes = FRM_SIZE;
QUANT_process( quant_comp,  buf_0, buf_1,
   &QUANTInArgs, &QUANTOutArgs );
HUFFInArgs.inBufSize = FRM_SIZE;
HUFFInArgs.outBufSize = FRM_SIZE;
HUFFInArgs.inBufValidBytes = FRM_SIZE;
HUFF_process( huff_comp,  buf_1, buf_0,
   &HUFFInArgs, &HUFFOutArgs );
// Output the bitstream
WriteBistream( &buf_0,
HUFFOutArgs.outBufValidBytes )
```

圖四　TI Codec Engine 中組合 codecs 的程式碼片段

```
// Server Configuration
var Server =
xdc.useModule('ti.sdo.ce.Server');
Server.threadAttrs.stackSize = 2048;
Server.threadAttrs.priority =
Server.MINPRI;
var defval = {stackSize: 4096, stackMemId:
0,priority: Server.MINPRI + 2}
Server.algs = [
   {name:"cst", mod: CST,
    threadAttrs: defval, groupId : 0},
   {name:"dct", mod: DCT,
    threadAttrs: defval, groupId : 0},
   {name:"quant", mod: QUANT,
    threadAttrs: defval, groupId : 0},
   {name:"huff", mod: HUFF,
    threadAttrs: defval, groupId : 0}, ];
```

圖五　Codec-Server 組態範例

```
//  Engine Configuration
var Engine =
xdc.useModule('ti.sdo.ce.Engine');
var myEngine = Engine.create("myengine", [
   {name: "cst", mod: CST, local: false},
   {name: "dct", mod: DCT, local: false},
   {name: "quant", mod: QUANT, local: false},
   {name: "huff", mod: HUFF, local: false},
]);
myEngine.server = "./all.x64P";
```

圖六　Engine 組態範例

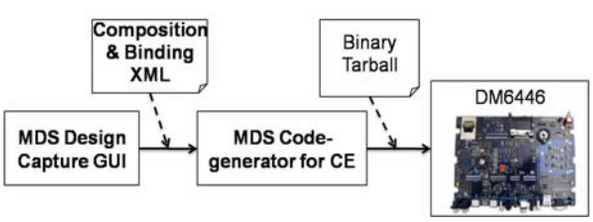
由上文中的描述，我們知道使用 TI Codec Engine 建立模塊化的應用程式，必須手動撰寫程式組合所有的 codecs，並且為元件間的通道加上緩衝區，這是很不直覺而且容易出錯的過程。雖然 TI 已經提供 XDC-Tool 協助 TI Codec Engine 應用程式的建置，但是其組態設定的複雜度卻讓一個初碰 TI Codec Engine 的新手難以領教。因此程式開發者使用 TI Code Engine 發展串流應用程式時，必須面對這些問題，而這些問題，有可能導致開發效率低落進而影響到上市的時程。

3. Multicore Development Suite

前一章節中，我們提到了 TI Codec Engine

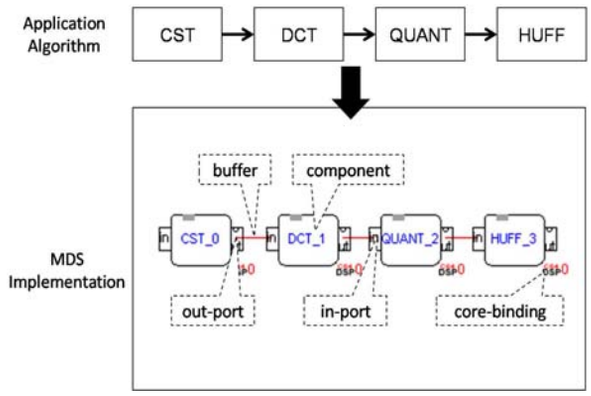
是一個很好的程式開發框架，讓程式開發者不用理會底層硬體的複雜運作。模塊化的軟體開發流程中，程式開發者以模型描述串流應用程式，每一個運算單元或元件稱作 codec，codec 與 codec 之間則以緩衝區連接，建立資料溝通管道；接著程式開發者把這個模型轉化成程式碼。過程中，除了元件與緩衝區的設定之外，還必須增加初始化與最後結束處理的程式碼。因此如果串流應用程式的模型常常在變動更改，將會造成程式開發者頻繁修改程式碼；甚至有可能因為人為的修改使得程式碼容易出錯。

MDS 的目的就是為了要解決這些問題。首先 MDS 提供了一個圖形化的串流應用程式模型建立工具 – MDS Design Capture GUI，讓程式開發者能用拖拉(drag-and-drop)的方式快速建立串流模型。接著 MDS 提供了另一個工具 – MDS Code-generator for CE，它能根據模型產生對應的 TI Codec Engine 應用程式。圖七是 MDS 所提出的設計流程。MDS Design Capture GUI 中建立的模型會被輸出成“Composition & Binding XML”，然後 MDS Code-generator for CE 把這個 XML 格式的中介檔讀入之後，產生能夠在 TI DaVinci DM6446 上執行的應用程式。我們將會在後面分開介紹這幾個工具。



(資料來源: Infonetics Research, July 2006)

圖七　MDS 所提出的設計流程



圖八　使用 MDS 進行串流應用程式的設計

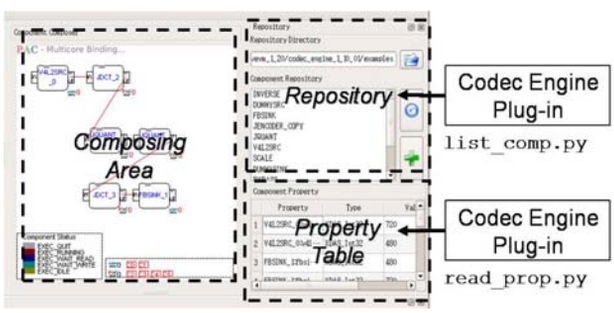
A. 串流應用程式模型圖形化編輯器 (MDS Design Capture GUI)

這個工具的目的是讓程式開發者以圖形化的方式，進行模塊化的串流應用程式設計。參考圖八，這裡所謂的串流應用程式是由**元件(component)**、**緩衝區(buffer)**、**輸出埠(out-port)**、**輸入埠(in-port)**、以及**運作核心(core-binding)**所組成。每一個元件都有獨立的名字與屬性設定。透過這幾個基本元素，我們可以用 MDS 實現圖四中的 JPEG 編碼器串流應用程式。使用這個工具的流程如下：

1. 從元件庫(repository)中抓出所有用得到的**元件**，並且把它們放到編輯區域(composing area)。
2. 使用**緩衝區**連結相關的**元件**。
3. 為每一個**元件**設定屬性。舉例來說，“quant”**元件**的 “quality”屬性，是用來設定量化的品質。
4. 設定**緩衝區**的大小。
5. 為每一個元件決定**運作核心**(元件會在指定的運作核心上執行)。
6. 最後將此串流模型存成一個 XML 檔案，用於自動

產生目標多核心平台的執行檔。此檔案格式將於下一個小章節進行介紹。

論文中，我們使用Qt[13]開發Design Capture GUI，將呈現與實作分離是設計這個工具的理念。這裡稱為呈現的部份是指視覺化表示串流模型的使用者介面(composing area)、元件庫的列表(repository)、以及屬性列表(property table)。因為Qt提供了豐富的使用者介面與功能強大的繪圖系統，我們利用Qt實現這個使用者介面，參考圖九左半部。至於實作的部份則是指使用者介面與TI Codec Engine整合，其中包含尋找適用於TI Codec Engine的codecs，並將它們列到元件庫的實作(透過呼叫list_comp.py擴充元件)，以及條列codec中所有屬性的實作(透過呼叫read_prop.py擴充元件)，參考圖九右半部。



圖九　MDS GUI 與其相關的擴充模組

在針對 TI Codec Engine 實作的擴充元件中，list_comp.py 以常規表示法搜尋特定目錄以取得所有可用的 codecs 的名字。其常規表示法與一個符合表示法的範例如下：

```
// regex: typedef\s+struct\s+I(\w+)_Fxns
typedef struct IV4L2SRC_Fxns {
   IALG_Fxns ialg;
   …
} IV4L2SRC_Fxns;
```

read_prop.py 則會於 codec 中的標頭檔裡尋找一個特定的結構以取得 codec 的所有屬性列表，其常規表示法與範例結構如下：

```
//regex: typedef\s+struct\s+I(\w+)_Params
typedef struct IV4L2SRC_Params {
   XDAS_Int32 v4l2srcWidth;  // Width
   XDAS_Int32 v4l2srcHeight; // Height
} IV4L2SRC_Params;
```

B. “Composition & Binding XML” 格式

由於 Qt 提供方便的 XML 分析工具，所以我們使用 XML 作為 MDS Design Capture GUI 與 MDS Code-generator 的溝通中介檔案，叫作“Composition & Binding XML”。 XML 裡所描述的元素有：元件、緩衝區、元件屬性、緩衝區屬性、各個元件對應運作核心的情形、以及一些圖形化介面上必須紀錄的屬性，如元件所在的 x,y 座標等等…。下面列了幾個範例：

這一段 XML 描述一個 JDCT 的元件已被建立，其名稱為 JDCT_2，id=2，並且被指派到第 0 個運算核心執行。在 Design Capture GUI 上，他的座標在(125,31)：

```
<Component classname="JDCT" name=”JDCT_2”
id="2" binding="0" X="125" Y="31"/>
```

這一段 XML 則描述一個 786432 bytes 大小的緩衝區，它連結了兩個元件，一個 id 為 0，另一個 id 為 2：

```
<Connection from="0" to="2" size="786432" />
```

至於屬性設定，下面是一個 JQUANT_4 元件的範例，描述它的 quality(量化品質)、isLuma(輸入資料是 luma 色頻或是 chroma 色頻)、isForward(正向量化或反向量化)屬性，每一

個屬性在 XML 中皆以一行表示：

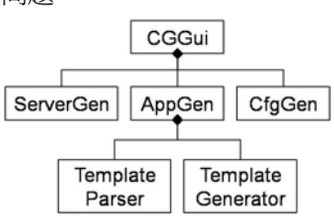
```
<Property component="JQUANT_4"
type="xdc_Float"
 value="1.0" name="quality" />
<Property component="JQUANT_4"
type="XDAS_Int32"
value="1" name="isLuma" />
<Property component="JQUANT_4"
type="XDAS_Int32"
 value="1" name="isForward" />
```

C. TI Codec Engine 串流應用程式碼產生器 (MDS Code-generator for CE)

MDS Code-generator for CE 的主要目的是產生直接就能夠在 TI DM6446 DVEVM 上執行的 TI Codec Engine 應用程式；此外還要幫助程式開發者處理建置應用程式的複雜流程。整個工具的架構如圖十所示，主要分成三個部份：Codec Server 產生器(ServerGen)、Engine 組態產生器(CfgGen)、與串流應用程式產生器(AppGen)。這三個部份剛好對應到圖二中所提 TI Codec Engine 的使用者角色 Role 2、Role 3、與 Role 4。接下來，我們會對三個工具分別作介紹：

1. Codec Server 產生器 (ServerGen)

ServerGen 是一個協助產生 Codec Server 的工具。根據“Composition & Binding XML”，它會自動把遠端執行的 codecs 加入 Codec Server 的組態檔。程式開發者不用擔心手動設定 Codec Server 常發生的問題，像是 codec 少放、多放、或是 codec 版本用錯等問題。



圖十　MDS Code-generator 的程式架構

2. Engine 組態產生器 (CfgGen)

CfgGen 是一個協助產生 TI Codec Engine 建置組態的工具。在本論文中，CfgGen 會產生兩種不同的組態：近端執行與遠端執行組態。如圖十一所示，建置的成果與組態設定有相關，近端執行的組態只會生成 ARM 端的執行檔；而遠端執行的組態則會生成 ARM 端執行檔與 DSP 端的二進制檔案(即 Codec Server)。CfgGen 會參考“Composition & Binding XML”中的 codec 執行方式的設定，產生遠端執行的組態；至於近端執行的組態，由於全部的 codecs 皆預設為近端執行，所以會忽視“Composition & Binding XML”中 codec 執行方式的設定，將所有 codec 安排在 ARM 上執行。

3. 串流應用程式產生器 (AppGen)

AppGen 負責產生對應到多核心平台上串流應用程式的實作，不過在產生的過程中，我們遇到輸入輸出裝置與 Codec Engine 的整合問題。典型的輸入輸出裝置像是檔案讀入元件、視訊輸入元件、檔案輸出元件、與圖像顯示元件。這類元件不適合直接實作成 TI Codec Engine 的 codecs，所以我們需要加入額外的程式碼實作這些輸入輸出元件，並且連結這些輸入輸出元件與串流應用程式。為了程式的靈活性，我們提出了一個以基於範本的程式碼產生引擎解決這個問題。圖十二是一個與檔案輸入輸出結合的範例，採用 TI Codec Engine 寫成的串流應用程式 – JPEG 編碼器在虛線的框框之內，由 file I/O 範本包裝組合成一個完整的串流應用程式。

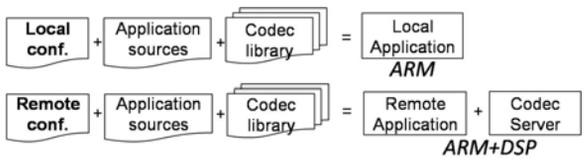
AppGen 的範本語法中定義了幾個特定的標

記，這裡稱為 MDS-Tag。表一中列了幾項 MDS-Tag 與其對應的串流應用程式碼片段。AppGen 會去分析範本的程式碼，在裡面尋找 MDS-Tag 出現的地方，然後把相對應的串流應用程式碼片段產生，取代 MDS-Tag 出現的位置。MDS-Tag 是以類似 C++樣板的格式訂立的。主要將產生的程式碼分成兩個類別：var 與 code。var 意思是這個 MDS-Tag 代表一個變數，有可能是字串或者是數字；code 則是代表這個 MDS-Tag 將會對應到一小段程式碼片段。圖十三是一個串流應用程式中排程器的程式碼片段範例。這個片段將會被插入到範本中{code_gen::code::sched_comps}出現的地方。

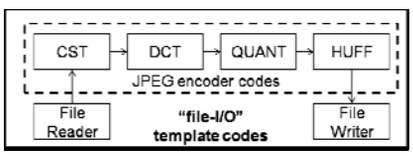
表一　PAC SoC 平台上的功率區塊整理

MDS-Tag	對應的程式碼片段
codec_gen::code::buf_decl	緩衝區的宣告程式片段
codec_gen::code::comp_decl	元件的宣告程式片段
codec_gen::code::comp_creation	實體化元件的程式片段
codec_gen::code::comp_set_params	元件屬性設定程式片段
codec_gen::code::alloc_buf	緩衝區記憶體分配程式片段
codec_gen::code::sched_comps	排程點程式片段

為了提升自動產生出來程式碼的穩定性，AppGen 會自動加入執行期錯誤處理的程式碼，像是例外處理、空指標測試、回傳值測試等等。AppGen 是一支高度模組化的應用程式，未來可以擴充串流應用程式最佳化的功能與進階錯誤處理的功能。這裡值得一提的是，錯誤處理並不完全是 AppGen 的責任，有些錯誤像是“ 靜態型別確認” ，可以直接由 MDS Design Capture GUI 支援。



圖十一　Local 與 remote 兩種組態設定的產出物圖示



圖十二　JPEG 編碼器的串流程式碼(虛線框內部)由 Code-generator 自動產生並嵌入“ file-I/O” 的範本之中，組合成一支完整的應用程式

D. TI Codec Engine codec 範本產生器 (CompGen)

程式開發者，在建立 codec 的時候，必須要加入一些與 xDAIS 介面有關，以及用於遠端程序呼叫的程式碼。每當新增一個 codec，這些程式碼都要重新撰寫，造成程式開發者不少困擾。除此之外，測試程式以及 codec 的組態檔，也要重新撰寫，非常容易造成人為上的錯誤。

有鑑於此，MDS 提供一個 codec 範本的產生器，叫作 CompGen，協助程式開發者快速建立一個能夠運作的 codec。並且產生相對應的測試程式。於是，程式開發者就不用在這些瑣事上花費額外的時間，直接就能專注於演算法的撰寫。而且也不用擔心過多的步驟，增加人為造成的錯誤。CompGen 也是一支基於範本的程式碼產生器，它輸出的目錄結構如表二所示，其結構參考自 TI Codec Engine 套件中之範例。

```
CSTInArgs.inBufSize = inBufSize;
CSTInArgs.outBufSize = bsize_0;
CSTInArgs.inBufValidBytes = inBufSize;
```

```
CST_process( comp_0,inBuf, buf_0, &CSTInArgs,
&CSTOutArgs );
DCTInArgs.inBufSize = bsize_0;
DCTInArgs.outBufSize = bsize_1;
DCTInArgs.inBufValidBytes =
CSTOutArgs.outBufValidBytes;
DCT_process( comp_1, buf_0, buf_1,
&DCTInArgs, &DCTOutArgs );
QUANTInArgs.inBufSize = bsize_1;
QUANTInArgs.outBufSize = bsize_2;
QUANTInArgs.inBufValidBytes =
DCTOutArgs.outBufValidBytes;
QUANT_process( comp_2, buf_1, buf_2,
&QUANTInArgs,
 &QUANTOutArgs);
HUFFInArgs.inBufSize = bsize_2;
HUFFInArgs.outBufSize = outBufMaxSize;
HUFFInArgs.inBufValidBytes =
QUANTOutArgs.outBufValidBytes;
HUFF_process( comp_3, buf_2, outBuf,
&HUFFInArgs, &HUFFOutArgs);
```

圖十三　MDS Code-generator for CE 產生之串流應用程式中“ 排程器程式片段”

表二　由 CompGen 所產生的目錄結

目錄	描述
apps/COMP_NAME	測試用程式
codecs/COMP_NAME	Codec，即元件的實作
extensions/COMP_NAME	Codec 於 TI Codec Engine 中支援遠端程序呼叫的程式碼

總結本章節的內容，目前在 MDS 中，除了 codec 的撰寫仍然需要手動之外，在元式式串流應用程式的開發與建置,程式開發者只須在 MDS Design Capture GUI 環境之中即可完成。

4. 實驗結果

本章節列出兩個實驗。第一個實驗目的是展示使用 MDS 所開發的串流應用程式與傳統手寫編輯應用程式相比，僅額外增加約 1%的效能損失。第二個實驗則展示如何使用 MDS 快速設定串流應用程式參數，用以觀察 TI DM6446 DVEVM 的遠端程序呼叫的額外效能損耗。

A. 使用 MDS 造成的效能損失 (MDS overheads)

程式開發者可能會懷疑，利用MDS的快速開發流程自動產生之串流應用程式，會造一些程式執行上的效能損失。為了比較傳統手動編輯的應用程式與MDS開發的串流應用程式效能上的差異，我們選擇IJG的JPEG編碼器 [14]與用MDS開發的JPEG編碼器(圖八)進行比較。我們拿512x512 大小ppm格式的影像lena作為輸入影像，比較手動編輯IJG的JPEG編碼器與兩個以MDS開發的JPEG編碼器三者的執行效能。表三中三個欄位代表不同的JPEG編碼器實作，分別是：IJG的JPEG編碼器實作(IJG)、MDS使用近端執行組態(MDS local conf) 的實作、MDS使用遠端執行組態(MDS remote conf) 的實作。垂直列中三個欄位為作業系統測量應用程式的執行時間，分別是：程式真實執行時間(Real)、在使用者空間中的執行時間(User)、在作業系統空間中的執行時間(System)。

表三　三種 JPEG 編碼器實作之效能比較 (單位：秒)

	IJG	MDS local conf.	MDS remote conf.
Real	0.282	0.284	0.264
User	0.260	0.250	0.120
Sys	0.020	0.030	0.080

IJG 的實作與 MDS local conf 實作都只使用 ARM 作為運算節點。在此實驗中，使用 MDS 開

發方式造成的效能損失約為 1%。這樣的效能損失在平常的應用程式中幾乎是可以忽略的。

為了展示使用 MDS 進行多核心程式開發所獲得的效益，我們比較了兩個不同組態的 MDS 應用程式，MDS local conf 與 MDS remote conf。使用者只需在 MDS 的開發環境中設定參數就可以利用到 DSP 的功能，並且得到 9%的效能增進。

B. 效能調校

調校多核心應用程式的關鍵在於使運算負載平衡與使資料流動順暢。在模塊化的軟體開發中，串流應用程式組成的元件數目與緩衝區的大小是兩個重要的參數。這裡我們固定一支應用程式 – JPEG 編碼器，利用 MDS 環境調整這兩個參數，並觀察其執行效能。

在 TI Codec Engine 的程式設計框架下，遠端程序呼叫所造成的額外效能損失也必須考量在內，而且這會與遠端程序呼叫的次數有關。在擁有相同功能的應用程式下，元件數目是和遠端程序呼叫的次數成正比的；而緩衝區的大小則會影響每次元件處理的資料個數，在同樣的資料量下，緩衝區愈小，元件就必須執行更多次，因此緩衝區的大小會和遠端程序呼叫的次數成反比。

為了驗證上述現象，我們重複使用實驗 A 中用 MDS 開發的 JPEG 編碼器。它是一個元件切割數目為四的應用程式，包含了四個元件：CST、DCT、QUANT、與 HUFF。我們又另外重新設計了一個獨立的 jpegenc 元件，並用它組成一個 JPEG 編碼器應用程式，它則是元件切割數目為一的應用程式。利用 MDS 圖形化工具我們可以快整建立兩者的模型，並且分別為他們調整緩衝區的大小，這裡設定兩個緩衝區大小：“Macroblock”與“Frame”。總共可以得到四種實驗組合。

表四為四個程式在 TI DM6446 DVEVM 上的實際執行時間。我們發現，功能完全相同的程式在不同緩衝區大小組態下的執行時間與預期相同。在緩衝區大小設定為 Macroblock 下，元件數目為 4 程式的執行時間又幾乎是元件數目為 1 程式的四倍。緩衝區大小設定為 Macroblock 的程式執行時間遠比緩衝區大小設定為 Frame 的程式執行時間要長到 5 倍以上。因此我們又去量測 TI DM6446 在執行遠端程序呼叫額外效能損失，分析結果如表五所示。在緩衝區大小為 Macroblock 的情況下，元件數目為 1 與 4 的程式中遠端程序呼叫的效能損失佔程式總執行時間的比率為 63.87%與 90.25%；而在緩衝區大小為 Frame 的情況下，卻為 2.74%與 4.99%。

這說明，在 TI Codec Engine 遠端程序呼叫的程式設計框架之下，緩衝區若太小(代表遠端程序呼叫的數目變多)或是元件數目太多(代表更多次的遠端程序呼叫)，會劇烈增加遠端程序呼叫的次數，因而造成嚴重的效能損失。但是若緩衝區的大小合理，元件的數目影響就不會太大，在這個狀況下，元件數目較多的應用程式，在多核心的平台上，有更多計算量配置上的彈性。利用 MDS 我們可以找到最適合的元件切割大小與緩衝區大小。

表四　由 MDS 組合之不同組態應用程式的執行時間

(單位：毫秒)

緩衝區大小 \ 元件數目	1	4
Macroblock (8×8×3)	1,290,905	4,201,821
Frame (512×512×3)	238,073	251,117

表五　由 MDS 組合之不同組態應用程式 RPC 執行效能損失的所佔總執行時間的比率

緩衝區大小 \ 元件數目	1	4
Macroblock (8×8×3)	63.87%	90.25%
Frame (512×512×3)	2.74%	4.99%

5. 結論

MDS 提供程式開發者一個視覺化的環境，讓使用者能夠以直覺化的方式開發多核心的串流應用程式；程式開發者在此開發環境中建立的應用程式可透過多核心平台程式產生器，直接產生能在多核心平台上執行的串流應用程式。

目前在 MDS 的軟體開發流程中，元件間目前只能採用固定大小的緩衝區做資料的傳遞，未來我們將會擴充緩衝區的功能，並加入特殊的模塊，支援 TLP 與 DLP。另外，我們預計新增多核心的視覺化除錯工具，希望能夠以模塊化軟體設計的觀念提供適合於串流應用程式的除錯方法。除此之外，還有整合程式碼層級上的除錯工具(如：GDB)，為程式開發者自動插入元件層級的中斷點，並自動對應到程式碼除錯工具上。我們預期 MDS 能協助多核心應用程式的開發者，快速進行多核心串流應用程式的開發，並且期待能夠開啟許多研究題目。

6. 參考文獻

[1] F. Poletti, et al, “Energy-efficient multiprocessor systems-on-chip for embedded computing: exploring programming models and their architectural support,” *IEEE Trans. Comput.*, vol. 56, pp.606-621, May 2007

[2] Simulink. The MathWorks, Inc. [Online]. Available: http://www.mathworks.com/products/simulink/

[3] NI LabView. National Instruments. [Online]. Available: http://www.ni.com/labview/

[4] DaVinci™ Technology Background and Specifications. Texas Instruments. [Online]. Available: http://focus.tij.co.jp/jp/lit/an/sprt401a/sprt401a.pdf

[5] Codec Engine Application Developer User’s Guide. Texas Instruments. [Online]. Available: http://focus.tij.co.jp/jp/lit/ug/sprue67d/sprue67d.pdf

[6] Codec Engine Server Integrator’s Guide. Texas Instruments. [Online]. Available: http://focus.tij.co.jp/jp/lit/ug/sprue67d/sprue67d.pdf

[7] Codec Engine Algorithm Creator User’s Guide. Texas Instruments. [Online]. Available: http://focus.tij.co.jp/jp/lit/ug/sprue67d/sprue67d.pdf

[8] TMS320 DSP Algorithm Standard Rules and Guidelines (Rev. G). Texas Instruments. [Online]. Available: http://www.ti.com/litv/pdf/spru352g

[9] TMS320 DSP Algorithm Standard API Reference (Rev. E). Texas Instruments. [Online]. Available: http://www.ti.com/litv/pdf/spru360e

[10] XDC Consumer User’s Guide. Texas Instrument. [Online]. Available: http://www.ti.com/litv/pdf/spruex4

[11] DSP/BIOS Kernel Technical Overview. Texas Instrument. [Online]. Available: http://www.ti.com/litv/pdf/spra780

[12] xDAIS-DM(Digital Media) User Guide. Texas Instrument. [Online]. Available: http://focus.ti.com/lit/ug/spruec8b/spruec8b.pdf

[13] Qt a cross-platform application framework. Trolltech, a Nokia company. [Online]. Available: http://trolltech.com/

[14] Free library for JPEG image compression.

Independent JPEG Group. [Online]. Available:
http://www.ijg.org/