# **Engineer-to-Engineer Note**

 Technical notes on using Analog Devices DSPs, processors and development tools

 Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or

 e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

## **Expert Code Generator for SHARC® Processors**

Contributed by Mitesh Moonat

Rev 5 – January 17, 2012

## Introduction

The Expert Code Generator (ECG) can be used to generate code for initializing specific SHARC® processor blocks to the system's requirements. The ADSP-214xx family of SHARC processors includes new modules and enhanced features compared to older SHARC processors. Some important new blocks include the Double Data Rate 2 (DDR2) controller (ADSP-2146x processors only), the Finite Impulse Response (FIR), Infinite Impulse Response (IIR), and Fast Fourier Transform (FFT) hardware accelerators. In addition, the ECG also produces initialization code for the Phase-Locked Loop (PLL), SDRAM, and Asynchronous Memory Interface (AMI) controllers both for new and the older SHARC processors (ADSP-2126x and ADSP-213xx processors). The Expert Code Generator consists of two utilities that can be directly integrated as plug-ins to VisualDSP++® development tools:

- Code generator for initializing PLL, DDR2/SDRAM, and AMI controllers
- Code generator for FIR, IIR, and FFT accelerators

To generate the code, you enter high-level system parameters without needing to know the processorspecific details. This saves software developers a lot of time and effort. This application note discusses, in detail, how to use these utilities to generate the required code.

## **Expert Code Generator Uses**

## Managing Limitations and Restrictions Automatically

Restrictions/limitations may need to be addressed when programming certain modules. One typical example is the maximum Voltage-Controlled Oscillator (VCO) frequency limitation for the PLL. This limitation implies that not all combinations of *PLLM* and *PLLD* of the PMCTL register can be used. You should ensure that the VCO frequency does not cross its maximum limit. Expert Code Generator eases the programmer's task by taking care of these types of restrictions.

# **Translating DDR2/SDRAM Device Specifications Taken Directly from the Device Data Sheet to the Code**

Initializing the DDR2/SDRAM controller for a particular DDR2/SDRAM device requires a number of control registers to be configured. Each timing specification needs to be programmed in terms of DDR2CLK/SDCLK cycles. Converting these timing specifications in terms of DDR2CLK/SDCLK cycles



requires significant effort and is prone to errors. Using Expert Code Generator, you need only to enter the required specification from the DDR2/SDRAM device data sheet; the tool takes care of everything else.

#### Following the Recommended Programming Sequence and Other Guidelines

To be able to program some modules, the programmer is expected to follow a specific sequence of instructions; e.g., programming the PLL must follow a set of recommended steps (refer to  $EE-290^{[1]}$ ), which if not followed, may result in unexpected results. With a number of modules being used in the system, it becomes difficult for the software developer to keep track each of such small recommendations and guidelines. Expert Code Generator takes care of these recommendations. Thus, it helps to reduce both the software development time and the chance of a system failure.

#### Handling IC Anomalies

IC anomalies requiring a software work around can be handled by the code generated by the tool itself.

#### Generating Code in Both "C" and "Assembly"

Expert Code Generator provides flexibility to the user to be able to generate code either in "C" or "assembly" language based on the application requirements.

## Code Generator for PLL, DDR2/SDRAM, and AMI Controllers

#### Overview

"Code Generator for PLL, DDR2/SDRAM, and AMI Controllers" helps to generate a source file with subroutine for initializing PLL, DDR2/SDRAM, and AMI controllers. The plug-in has options to select the processor and the corresponding speed grade for which the code has to be generated. It shows only those sections out of PLL, DDR2, SDRAM, and AMI controllers which are supported by the processor selected. E.g Figure 1 shows the snapshot of the plug-in for the ADSP-21262 processor. It consists of single section for initializing the PLL. Figure 2 shows a snapshot of the plug-in for the ADSP-21369 processor. It consists of three sections corresponding to the PLL, SDRAM, and AMI controllers. Figure 3 shows snapshot of the plug-in for ADSP-21469 processor. It consists of three sections corresponding to PLL, DDR2, and AMI controllers. Each section allows you to enter/select values for a set of parameters needed to initialize the corresponding module. Appropriate error messages display when you enter an out-of-range value or if an entered value may indirectly cause an invalid result. Furthermore, it provides the option to save the current system configuration in a .CFG file, which can be used later to restore the same configuration. One can as well bring all the settings back to the default values by clicking on the Reset button.



PLL, DDR2/SD	RAM, and AMI I	nitialization Code	e Generato 🔀			
PLL Initializat	ion					
CLKIN	25	VCOmax	800			
CCLK	200					
MAX CCLK	200					
MIN CCLK	0					
Actual Values that will be generated CCLK 200.000000 MHz						
Code Generation Language Select Module Assembly C PLL Generate Code ADSP-21262 200 200 200 200 200 200 200 200 200						
System Configuration Save Load Reset						

Figure 1. Code generator for PLL of the ADSP-21262 processor

PLL, DDR2/	5DRAM, a	nd AMI I	nitialization Co	de Generato	r (Rev 5)						×	
PLL Initiali	zation				SDRAM Control	ller Initialization –			AMI Controller Initializa	ation		
CLKIN	24.576		VCOmax	800	tRAS (ns)	42	Which Bank?	MS2 🔽	Select Bank	MS0 🗸	1	
CCLK	400		SDCLK:CCLK	2.5 🗸	tRP (ns)	18	CAW	8 🗸	_			
MAX CCL	K 400				ťWR (ns)	14	RAW	12 🗸	Enable AMI Bank		-	
MIN CCL	< 0				tPCD (pp)	10	SDCL	3 💙	Wait States	31 🗸	1	
					IRCD (IIS)		Optimize Reads	<b>○</b> Y <b>⊙</b> N	Packing Order	LSBF 🗸		
					tREF (ms)	64	Read Modifier	1 🗸	Read Hold Cycles	0		
Actual	/alues that	will be ger	nerated		NRA	4096	Power Up Mode	REF, MR 🔽	Write Hold Cycles	0		
CCLI	CCLK 393.216003 MHz			PGSZ 128 ?	OY ON	Bus Width	32 💙	Bus Midth	0			
SDC	-K	157.2864	.01	MHZ	Burst Stop		ADDRMODE	Bank inte 🗸	Duo main	•		
					Disable ?	OY UN	-Include Subrou	itines to	Enable ACK Sign	nal		
Error M	essade					Enter Self Refresh				Disable Predictive Reads		
							Exit S	elt Retresh	Disable Packing			
Code Ger	eration —								Apply			
						s selected for MT	- international and internatio					
0 400	Processor Speed grade					009 EZ-NIL						
Generate	Code	ADSP-213	369 🔽	400	]							
System C	onfiguration	n										
		Save	Load	Reset								

Figure 2. Code generator for PLL, SDRAM, and AMI controllers of the ADSP-21369 processor



LL, DDR2/SD	RAM, a	nd AMI I	nitialization Co	de Generator	(Rev 5)									
PLL Initializat	tion —				DDR2 Controller	Initialization —				AMI Controller Initializa	ation			
CLKIN	25		VCOmax	900	tRAS (ns)	40	Which Bank?	CS0	-	Select Bank	MS0	~		
CCLK	450		DDCLK:CCLK	2 🗸	tRP (ns)	15	Banks	8	-	_				
MAX CCLK	450		MLBCLK:CCLK	1:4 🔽	t/A/TR (ps)	7.5	CAW	10	-	Enable AMI Bank				
MIN CCLK	0		LPCLK:CCLK	1:3 💙		с	RAW	13	-	Wait States	31	~		
First Time	e progra	mming			tRCD (ns)	15	DDCL	4	-	Packing Order	LSBF	~		
CLK_CFG	x 1	6:1 🗸	fVCOcurrent	800	tRTP (ns)	7.5	DDAL	0	-	Read Hold Cycles	0	~		
Actual Val	lues that	will be ge	nerated		tRRD (ns) 10 ODT (Rtt) Disabled V		-	Write Hold Cycles	0	~				
CCLK		450.0000	00	MHz		50	DQS Enable	Enabled	-	Idle Cycles	0	*		
DDR2C	LK	225.0000	00	MHz	LFAVV (IIS)	50	Optimize Reads	Optimize Reads OY ON		Bus Width	8	~		
MLBCL	к	112.5000	00	MHz	tWR (ns)	15	Read Modifier 1			Enable ACK Signal     Disable Predictive Reads				
LPCLK		150.0000	00	MHz	tREFI (us)	7.8								
-Error Mess	sage				tRFC (ns)	127.5	Exit Self Refresh							
					max tCK (ns)	8	Enter Power Down							
Code Gener	ation —	Se	ect Module				Exit Po	wer Down		Apply				
Assembly C PLL DDR2 AMI				Default values device on ADS	s selected for t SP-21469 EZ-Ki	he MT47H64M16 D t								
Processor Speed grade														
Generate Code ADSP-21469 V 450 V														
System Configuration														
	Save Load Reset													

Figure 3. Code generator for PLL, DDR2, and AMI controllers of the 21469 processor

#### **Registering and Accessing the Plug-In**

To integrate the plug-in with the VisualDSP++ environment, perform the following steps:

- 1. Copy the PLL\_DDR2\_SDRAM\_AMI\_Init.dll file to the <install\_path>\VisualDSP 5.0\System folder.
- 2. Copy the regsvr32.exe file from C:\WINDOWS\System32 to the folder mentioned above if this file is already not available.
- 3. From a CMD prompt, go to the above folder and register the .dll file as shown below: <install\_path>\VisualDSP 5.0\System> regsvr32.exe PLL\_DDR2\_SDRAM\_AMI\_Init.dll



Figure 4. Registering "Code Generator for PLL, DDR2/SDRAM, and AMI Controllers"



For Windows Vista® and Windows 7® operating system, the command prompt should be opened in administrator mode as shown in Figure 5.

Command Pre	mnt		Recent Items				
📧 Connect to a l		Open					
Notepad	۲	Run as administrato	r				
🔏 Paint		Open file location					
🕵 Remote Deskt	ą	WinZip	•				
🖅 Run	- 570	Pin to Start Menu					
Snipping Tool		Add to Quick Launch					
C Sound Record		Add to Quick Edulic					
Sync Center		Restore previous ver	sions				
Welcome Cen		C 17					
Windows Expl		Send To	· · · ·	ALV.			
Windows Mol		Cut		ort			
Windows Side		Copy					
WordPad		.,					
Ease of Acces		Delete					
System Tools		Rename					
Jablet PC		Deservation					
Windows Pow		Properties					

Figure 5. Accessing command prompt as administrator in Windows Vista OS

4. Once the registration is completed, the plug-in can be accessed via: Tools->Plugins->PLL, DDR2/SDRAM, and AMI Initialization Code Generator as shown in Figure 6.

Analog Devices VisualDSP++ - [Target: ADSP-21479 ADSP-2	14xx Simulator]	
File Edit Session View Project Register Memory Debug Settings	Tools Window Help	
💽 🗅 🖨 🖬 🖉 🍧 💥 👹 👗 🖻 🖻 🗠 🖂 🕅	Trace	\$ % % 🐐 🖗 😵
	Linear Profiling	R R. R.
	Expert Linker	
目 🗊 👃 🕘 🧶 💹 💹 🔁 🖓 🖓 🖓 🖓 🖓 👰 🚳	Elash Programmer	
•	PGO •	
	<u>Plugins</u>	Accelerators
		PLL, DDR2/SDRAM, and AMI Initialization Code Generator

Figure 6. Accessing "Code Generator for PLL, DDR2, and AMI Controllers"

#### **Processor and Speed Grade Selection**

The first step to be able to generate the code using this plug-in is to select the processor and the corresponding speed grade. As discussed above, depending upon the processor selection, one or more sections out of PLL, DDR2, SDRAM, AMI controllers shall be activated. These sections will again show only the fields supported by the selected processor. E.g. in the PLL Initialization section for



ADSP-21262 processors, the fields corresponding to DDR2CLK, MLBCLK, and LPCLK are not shown unlike for ADSP-21469 processors.

#### PLL Initialization

Perform the following steps to include PLL initialization code:

- 1. In CLKIN, enter a value in MHz. By default, this field is initialized with the CLKIN frequency used in the EZ-KIT® evaluation board corresponding to the selected SHARC processor. E.g. default CLKIN value for ADSP-2147x processors is 16.625 MHz as used in the ADSP-21479 EZ-KIT board.
- 2. In CCLK, enter the required value in MHz. By default, this value will be equal to the maximum value supported by the processor of a particular speed grade. The exact value of the required CCLK may not be achieved in some cases. For such cases, enter the minimum (MIN CCLK) and maximum (MAX CCLK) CCLK range in MHz. The tool will generate the closest possible CCLK to the CCLK entry within the given range. By default, MIN CCLK = 0, and MAX CCLK = maximum supported CCLK.
- 3. The field VCOmax shows the maximum supported VCO frequency by the combination of selected processor and speed grade.
- 4. Depending upon the processor selected, select various ratios such as DDR2CLK/SDCLK:CCLK, LPCLK:CCLK etc.
- 5. For 214xx processors, check/uncheck "First Time programming" depending upon whether the PLL programming macros will be used for programming the PLL first time after reset or not. If this option is unchecked, another option "INDIV before programming" will appear. Check "0" or "1" depending upon whether INDIV is set or cleared before the PLL programming. These options are required to take care of the PLL anomaly#15000020.
- 6. Under Select Module, select the PLL check box.

The Actual Values that will be generated group box displays the actual CCLK value that is the closest possible value to the required CCLK value to be generated by the tool. It also displays the resulting DDR2CLK/SDCLK, MLBCLK, and LPCLK values based on this actual CCLK value and the corresponding selected ratios. "*Error*!" will be displayed for all the clocks if there is an error in CCLK generation. When there is no error in CCLK generation but one or more of DDR2CCLK/SDCLK, MLBCLK, or LPCLK are out of the supported range, "*Error*!" will be displayed in front of that clock. View the text displayed in the Error Message box to see the message that corresponds to a particular error. No text will be displayed if there is no error. Figure 7 shows examples of error conditions.



PLL, DDR2/SD	and AMI I	nitialization Co	le Gene	PL	L, DDR2/SD	RAM, a	nd AMI I	nitialization Co	ode Generator			
PLL Initialization							PLL Initializat	tion				
CLKIN	25		VCOmax	900		L	CLKIN	25		VCOmax	900	
CCLK	460		DDCLK:CCLK			L	CCLK	450		DDCLK:CCLK	4 🗸	
MAX CCLK	450		MLBCLK:CCLK	1:4	~	L	MAX CCLK	450		MLBCLK:CCL	<b>K</b> 1:4 🖌	
MIN CCLK	0		LPCLK:CCLK	1:3	*	L	MIN CCLK	0		LPCLK:CCLK	1:3 👻	
- Actual Val	lues that	t will be ge	nerated			L	- Actual Val	ues that	will be ge	nerated		
CCLK		Error!		MHz		L	CCLK		450.0000	00	MHz	
DDR2C	LK	Error!		MHz		L	DDR2C	LK	112.500000>Error!		MHz	
MLBCL	к	Error!		MHz		L	MLBCLK		112.500000		MHz	
LPCLK		Error!		MHz		L	LPCLK		150.0000	0000 MHz		
CCLK more	sage e than th	ne maximur	n supported				Error Mess	sage < minimu	um specifie	d value		
Code Gener Language O Assem	Code Generation Language Select Module DDR2 Assembly C Processor Speed grade Frocessor Speed grade Frocessor Speed grade Frocessor Speed grade Frocessor Speed grade State Sta						Code Generate Co	ation — bly O ode	C	elect Module	R2 AMI	
System Configuration Save Load Reset							- System Con	figuratio	in S	ave Load	Reset	

Figure 7. Various error conditions

The plug-in does not include the workaround for anomaly number "15000014" on ADSP-214xx SHARC processors as it is applicable only for the DMA mode. If the system uses MLB in DMA mode, the workaround has to be added explicitly. Please refer to the corresponding anomaly sheet for more details<sup>[10][11][12]</sup>.

#### **Initializing the DDR2 Controller**

Perform the following steps to include DDR2 controller initialization code:

- 1. Enter the following specifications from the DDR2 device data sheet:
  - □ tRAS
  - □ tRP
  - □ tWTR
  - □ tRCD
  - □ tRTP
  - □ tRRD
  - □ tFAW
  - □ tWR
  - □ tREFI
  - □ tRFC
  - □ tCK max (minimum DDR2 clock frequency supported by the device)
  - □ Banks (number of banks)



- □ CAW (column address width)
- **RAW** (row address width)
- DDCL (CAS latency)
- **DDAL** (additive latency)
- 2. In ODT (Rtt), Select Disabled to disable on-die termination, or select the corresponding value for Rtt.
- 3. In DQS Enable, select Enabled or Disabled according to the system's requirements.
- 4. In Optimize Reads and Read Modifier, select Yes if predictive reads are needed, and select the corresponding modifier value.
- 5. In Which Bank, select the external port bank to which the DDR2 device should be mapped.
- 6. In Include Subroutines to, select options to generate separate subroutines for entering and exiting Self Refresh and Power Down modes. Corresponding options should be checked to generate these subroutines along with the PLL\_DDR2\_AMI subroutine.
- 7. Under Select Module, select the DDR2 check box.

#### **Initializing the SDRAM Controller**

- 1. Enter the following specifications from the SDRAM device data sheet:
  - □ tRAS
  - □ tRP
  - □ tWR
  - □ tRCD
  - **u** tREF(SDRAM refresh period)
  - □ NRA (number of row addresses)
  - □ CAW (column address width)
  - **RAW** (row address width)
  - □ SDCL (CAS latency)
- 2. For ADSP-2137x, ADSP-2147x, and ADSP-2148x processors, select Yes for PGSZ 128 to set column width to 7 bits.
- 3. For ADSP-2137x processors, select Yes for Burst Stop Disable to disable burst stop.
- 4. In Optimize Reads and Read Modifier, select Yes if predictive reads are needed, and select the corresponding modifier value.
- 5. In Power Up Mode, select REF, MR Set for the power up mode "Precharge, mode reg set, 8CBR refresh cycles" else select MR Set, REF for the power up mode "Precharge, 8CBR refresh cycles, mode reg set".
- 6. In Bus Width, select the bus width to be either 32 or 16 bits as per system requirements.



- 7. In Which Bank, select the external port bank to which the SDRAM should be mapped.
- 8. For ADSP-2147x and ADSP-2148x processors, in ADDRMODE, select the addressing mode to be page or bank interleaved.
- 9. In Include Subroutines to, select options to generate separate subroutines for entering and exiting Self Refresh mode.
- 10. Under Select Module, select the SDRAM check box.

#### **Initializing the AMI Controller**

The AMI Controller Initialization section of the GUI allows you to enable and configure one or more of the four AMI banks individually. To configure any AMI bank and include the AMI initialization code, perform the following steps:

- 1. In Select Bank, select the bank to be configured.
- 2. Select Enable AMI Bank, to enable this AMI bank.
- 3. Select the following parameters for this bank:
  - □ Wait States
  - □ Packing Order
  - Read Hold Cycles
  - □ Write Hold Cycles
  - □ Idle Cycles
  - **D** Bus Width
  - Enable/Disable Packing
  - **D** Enable/Disable Predictive Reads
  - **D** Enable/Disable ACK Signal usage
- 4. Click Apply to save the settings for the current AMI bank selected.
- 5. Under Select Module, select the AMI check box.

#### **Generating the Code**

Once all the settings are entered and the required modules are selected, perform the following steps:

- 1. Under Language, select the Assembly button to generate the code in assembly, or select the c button to generate the code in C.
- 2. Click the Generate Code button to save the generated file in the required project folder. The default file name would be PLL\_Init/PLL\_SDRAM\_AMI\_Init/PLL\_DDR2\_AMI\_Init.asm (or C) depending upon the processor selected.



#### **Using the Generated Source File**

Perform the following steps to use the generated source file in a VisualDSP++ project:

- 1. Add the generated file to the project.
- 2. In Assembly, declare the function \_Init\_PLL/\_Init\_PLL\_DDR2\_AMI/\_Init\_PLL\_SDRAM\_AMI.asm external using the .extern directive. Call the function whenever needed in the main code.
- 3. In *C*, declare the function Init\_PLL()/Init\_PLL\_SDRAM\_AMI()/Init\_PLL\_DDR2\_AMI()external using the extern directive; Call the function whenever needed in the main code.

Figure 8 shows PLL\_DDR2\_AMI\_Init.asm generated for the default settings for ADSP-21469 processor. As shown, the code in the source files calls macros such as ADSP\_2146x\_PLL\_Init, ADSP-2146x\_DDR2\_Init, and ADSP-2146x\_AMI\_Init. These macros are defined in the file cs\_macros.h supplied with this tool.

Ensure that the file cs\_macros.h is copied to the <install\_path>\VisualDSP 5.0\<212xx/213xx/214xx>\include folder.

Figure 9 shows an example VisualDSP++ project PLL\_DDR2\_AMI\_Init in assembly language using the source file PLL\_DDR2\_AMI\_Init.asm.



Figure 8. PLL\_DDR2\_AMI\_Init.asm



Analog Devices VisualDSP++ - [Targe	t: ADSP-21469 via HPPCI-ICE] - [Project: PLL_DDR2_AMI_Init] - [main.asm]	
File Edit Session View Project Registe	ar Memory Debug Settings Tools Window Help	
🔝 🗅 🛩 🖬 🖉 🍏 💥 😽 🛛	K 🖻 🛍 🗠 🕾 🗚 🎢 🐗 🍓 🌮 🕮 🥢 🌤 🌤 🧏 🕌 🙀 😵	
+0 ×   Pa Pa Pa   🕸 🕮 📇 🍝  [	PLL_DDR2_AMI_Init 🔽 Debug 💽 🚺 📭 📭	
i 🗉 🚿 🍐 🤚 🕾 🙊 🕅 🛃 i i i i	- (P- 40) 667 💭 🖗 💭 📄 🖻 🗂 🖶 🗊 😰	
Project: PLL_DDR2_AMI_Init.dp)	<pre>#include<def21469.h> .global _main; .extern _Init_PLL_DDR2_AMI; .section/pm seg_pmco; _main:     call _Init_PLL_DDR2_AMI;     r0=0xaaaaaaa;     dm(0x200001)=r0;     r1=dm(0x200001); </def21469.h></pre>	
	0420001         R0 04AAAAAA00         R8 000B21B500           0x200001         R0 0AAAAAAA00         R9 000000000           [200001] AAAAAAAA FBDFEEFF         R0 000000000         R1 06583E74900           [200001] FFFFFFFF 575BDFF         R0 000000000         R1 06583E74900           [200007] FFFBFFF 575BDFF         R0 000000000         R1 06583E74900           [200007] FFFBFFF 575BDFF         R0 000000000         R1 0008022A00           [200008] FFFFFFFF 7F7EFFFF         R0 0000000000         R1 0008022A00           [200008] FFFFFFF 7F7EFFFF         R0 000000000         R1 0008000000           [200008] FFFFFFFF 7FFFFFF         R0 000000000         R1 000000000           [200008] FFFFF7FFF         7FFEFFFF         R7 0000000000         R1 819A68A000           [200008] FFFF7F7FFFFFF         7FFEFFFF         R7 0000000000         R1 5 0000498000           [200008] FFFF77FFFFFFFFF         7FFFFFFF         70000000000         R1 5 0000498000           [200001] D7FFFD7F FFFFFFFF         7         7         7         7	

Figure 9. VisualDSP project using the GUI Generated source file "PLL\_DDR2\_AMI\_Init.asm"

Figure 10 shows a snapshot of the PLL\_DDR2\_AMI.C file generated for the default settings.

Figure 11 shows an example VisualDSP++ project PLL\_DDR2\_AMI\_Init\_C in C language using the source file PLL\_DDR2\_AMI\_Init.C.



Figure 10. PLL\_DDR2\_AMI\_Init.C



Analog Devices VisualDSP++ - [Target:	: ADSP-21469 via HPPCI-ICE] - [Project: PLL_DDR2_AMI_Init_C] - [main.C]						
File Edit Session View Project Register	Memory Debug Settings Tools Window Help						
🔊 🛮 🗅 😂 🔜 🕼 🎒 🐇 🖏 📲 🌾 🗠 🗠 🛤 🎢 💠 🐜 🌮 🗐 🥕 🎋 🧏 🦌 🦓 😵							
🗕 🛪   🖳 🍡 🗞   🇇 🛗 🚟 🍊   Pl	LL_DDR2_AMI_Init_C 🔽 Debug 💽 💽 🚱 🚱						
	(የ**() 66' 💭 🖗 💭 📄 🗖 🖬 🖆						
Project: PLL_DDR2_AMI_Init_C.dp  Project Group (1 project)  PLL_DDR2_AMI_Init_C*  Main.C  Linker Files Header Files	<pre>     #include<def21469.h>     #include<cdef21469.h>     extern void Init_PLL_DDR2_AMI();     int temp_write=0xaaaaaaaa;     int temp_read;     int* temp=(int*)0x200001;     int main()         Init_PLL_DDR2_AMI();         *temp=temp_write;         temp_read=*temp;     } } </cdef21469.h></def21469.h></pre>						
	• return 0;						
	Data(DM) Memory [Hexadecimal] Data(DM) Memory [Hexadecimal]						
	Image: State of the state o						

Figure 11. VisualDSP++ project using the GUI-generated source file "PLL\_DDR2\_AMI\_Init.C"

## **Code Generator for Accelerators**

#### Introduction

"Code Generator for Accelerators" generates code for using FIR, IIR, and FFT hardware accelerators on ADSP-214xx processors. It generates a single file named FIR\_IIR\_FFT\_Init.asm or FIR\_IIR\_FFT\_Init.C, which includes TCB declarations and subroutines for configuring and initializing FIR, IIR, and FFT accelerators. Appropriate error messages are displayed in response to an invalid value entered by the user. It also shows the on-chip memory that will be used to store for input/output/coefficient buffers and the TCBs. Furthermore, it provides an option to save the current system configuration in a .CFG file, which can be used later to restore the same configuration. One can as well bring all the settings back to the default values by clicking on the Reset button.



Code Generator for A	Accelera	ators												×
- FIR Accelerator-	all chann	nels				- IIR Accelerator	r all channe	els			-FFT	Accelerator - eneral Settings	8	]
No of Channels	1	Data	Format	Floating 🗸	1	No of Channels	1 🗸	Data	a Format	32 bit Floating 🗸	N	lo. of Points	16	~
Auto Iterate ?	No 🔹	Round	ling Mode	Nearest(ever 🗸		Auto Iterate ?	No 🗸	Roun	ding Mode	Nearest(ever 🗸	In	nput Packing	R0,R1I0,I1.	~
DMA Interrupt	None 🔹	<ul> <li>Status</li> </ul>	s Interrupt	None 🗸		DMA Interrupt	None 🗸	Statu	s Interrupt	None 🗸	01	utput Packing	R0,R1I0,I1	~
Interrupt When	All Cha	innels Con	nplete 🗸			Interrupt when	All Chann	els Cor	nplete 🗸			Repeat?	OYes 📀	No
Channel Specific S	ettings -					Channel Specific !	Settings				D	MA Interrupt	None	*
Select Channel	1	~	-Buffers-						Buffers-		Sta	atus Interrupt	None	*
	_		Select But	ffer Input 🗸		Calent Observe			Select But	ffer Input 🗸	GBu	ffers		
Tap Length	512					Select Channe	1	*			Se	lect Buffer	nput	~
Window Size	512		Base	FIR_IP_buff1					Base	IIR_IP_buff1		Name	EET ID huff	_
Sample Rate Cor	iversion		Index of	fset 0		No. of Biquads	: 1	~	Index Off	iset 0		I south	32	
Enable			Modifier	1		Window Size	512		Modifier	1		Length	-	
Interpolation			Length	512		,			Length	512		Save Buffer	Settings	
Ratio			Save But	ffer Settinge		Save Chann	el Settings		Save B	uffer Settings	On	Chip Memory	/Usage	$\equiv$
			5470 04	ner octangs							тс	Bs	576 Bits	
Save Channel Se	ettings					TCBs 416 Bit	sage	Out	out	40.000000.1/5	Inn	ut	1 000000 8	Ch
On Chip Memory Us	sage					10.000000 Kb								Ĩ
TCBs 416 Bits Output 16.000000 Kb											itput	1.0000001	(b	
Input 16.000000 Kb Coefficient 16.000000 Kb					Error Message					Co	efficients	17.000000	Kb	
Error Message Pointwise Twiddle NA														
	IM Outout NA													
System Configuratio						- Code Generation -			Include	Code for				
		Save	Load	I Reset		Generate Code	Assembl	y O C	FIR	IR FFT	Proce	essor AD	SP-21469	~

Figure 12. Code generator for Accelerators

### **Registering and Accessing the Plug-In**

- 1. Copy the file ACC\_Init.dll to the <install\_path>\VisualDSP 5.0\System folder.
- 2. Copy regsvr32.exe file from C:\WINDOWS\System32 to the folder mentioned above (if it's already not available).
- 3. From a CMD prompt, go to the above folder and register the .dll file as shown below:

<install\_path>\VisualDSP 5.0\System> regsvr32.exe ACC\_Init.dll



For Windows Vista® and Windows 7® operating system, the command prompt should be opened in administrator mode as shown in Figure 5.



-
-

Figure 13. Registering "Code Generator for Accelerators"

4. After registration is completed, the plug-in can be accessed via: Tools->Plugins->Accelerators as shown in Figure 14.



Figure 14. Accessing "Code Generator for Accelerators"

#### **FIR Accelerator**

This section describes how to generate TCBs and the initialization code for the FIR accelerator. You are responsible for selecting both the global and channel-specific parameters. For multichannel processing, configure multiple sets of input, output, and coefficient buffers and individual channel-specific parameters for each channel separately. Perform the following steps to generate code for FIR accelerator:

- 1. Specify the following global settings applicable to all channels:
  - No. of channels: Number of channels chained to be processed one after the other.
  - □ Data Format: Type of data that is to be processed: 32 bit Signed/Unsigned Fixed Point, or 32 bit Floating Point.
  - □ Auto Iterate: Specifies whether the processing should stop after all channels are processed.
  - Rounding Mode: For floating-point operation. Various rounding modes can be selected, e.g., IEEE round to nearest, IEEE round to zero, Round away from zero, etc.



- □ Interrupt when: Interrupt can be generated either when processing of all channels or each channel is complete.
- □ DMA Interrupt: "None" indicates that the DMA interrupt is not being used in the application. Select any of the 19 programmable interrupts from "POI" to "P18I" to which this interrupt should be mapped.
- □ Status Interrupt: "None" indicates that the DMA interrupt is not being used in the application. Select any of the 19 programmable interrupts from "POI" to "P18I" to which this interrupt should be mapped.
- 2. Configure channel-specific settings:
  - □ Select Channel: Select the channel to be configured.
  - □ Tap Length: Enter the tap length for the FIR filter (<= 4096).
  - □ Window Size: Enter the number of output samples to be generated for one iteration (<= 1024).
  - □ Sample Rate Conversion:

Select Enable if sample rate conversion is required.

Select whether Decimation or Interpolation is required.

Select the decimation/interpolation ratio.

- 3. Buffers: Configure following settings for the input/output/coefficient buffers:
  - □ Select Buffer: Select input/output/coefficient buffer for which setting is required.
  - **D** Base: Default name is FIR\_IP\_Buffx/ FIR\_OP\_Buffx/ FIR\_CF\_Buffx (x = channel number). Enter a new name if the name of the buffer is different.
  - □ Index Offset: Enter the offset from where the processing needs to be started with regard to the base address.
  - □ Enter Modifier and Length.
  - □ Changing any field under Buffers enables the Save Buffer Settings button. Click this button to save the new settings. The button will be disabled automatically after it is clicked.
  - □ Changing any field under Channel Specific Settings (except Buffers) enables the Save Channel Settings button. Click this button to save the new settings. The button will be disabled automatically after it is clicked.
- 4. Under Include Code for, select the FIR check box.



#### **IIR Accelerator**

This section describes how to generate TCBs and the initialization code for the IIR accelerator. You are responsible for selecting both the global and channel-specific parameters. For multichannel processing, configure multiple sets of input, output, and coefficient buffers and individual channel-specific parameters for each channel separately. Perform the following steps to generate code for IIR accelerator:

- 1. Select following global settings applicable to all channels:
  - No. of channels: Number of channels chained to be processed one after the other.
  - □ Data Format: Type of data to be processed: 32/40 bit Floating Point.
  - □ Auto Iterate: Specifies whether the processing should stop after all channels are processed.
  - Rounding Mode: For floating-point operation. Various rounding modes can be selected, e.g., IEEE round to nearest, IEEE round to zero, Round away from zero, etc.
  - □ Interrupt when: Specifies whether an interrupt is generated when processing all channels or when each channel is complete.
  - □ DMA Interrupt: "None" indicates that the DMA interrupt is not being used in the application. Select any of the 19 programmable interrupts from "POI" to "P18I" to which this interrupt should be mapped.
  - □ Status Interrupt: "None" indicates that the status interrupt is not being used in the application. Select any of the 19 programmable interrupts from "POI" to "P18I" to which this interrupt should be mapped.
- 2. Configure channel-specific settings:
  - Select Channel: Specifies the channel to be configured. The total number of channels that needs to be configured will depend on the No. of channels value selected in the global settings. The total number of channels in the Select Channel option will be dynamically updated based on the No. of channels setting.
  - No of Biquads: Enter number of biquads for the IIR filter (<= 12).
  - □ Window Size: Enter the number of output samples to be generated for one iteration (<= 1024).
  - □ Buffers: Configure following settings for the input/output/coefficient buffers.

Select Buffer: Select Input/Output/Coefficient buffer for which setting is required.

Base: The default name is IIR\_IP\_Buffx/ IIR\_OP\_Buffx/ IIR\_CF\_Buffx (x = channel number). Enter a new name if the name of the buffer is different.

Index Offset: Enter the offset from where the processing needs to be started with regard to the base address.

Enter a Modifier and a Length.

Changing any field under Buffers enables the Save Buffer Settings button. Click this button to save the new settings. The button will be disabled automatically after it is clicked.



Changing any field under Channel Specific Settings (except Buffers) enables the Save Channel Settings button. Click this button to save the new settings. The button will be disabled automatically after it is clicked.

3. Under Include Code for, select the IIR check box.

#### **FFT Accelerator**

This section describes how to include TCBs and initialization code for FFT accelerator. Perform the following steps to include the FFT accelerator code:

- 1. General Settings:
  - No. of Points: Specify the number of FFT points required (2<sup>k</sup>, 4<=k<=13).
  - □ Input Packing and Output Packing: Select the input and output packing format.

Select "*R0*, *R1...10*, *I1..*" when input/output data is required in "first all real, then all imaginary" format.

Select "R0, I0, R1, I1.." when input/output data is required in "alternate real and imaginary" format.

**NOTE**: For FFT points>256, the packing format setting is ignored (i.e., input and output both use "alternate real and imaginary" format).

**D** Repeat: Specify whether the processing needs to be repeated when done.

**NOTE**: You may have to add extra part of the code based on the application requirements when repeat mode is selected.

- □ DMA Interrupt: Select "None" to indicate that the DMA interrupt is not being used in the application, or select any of the 19 programmable interrupts from "*P0I*" to "*P18I*" to which this interrupt should be mapped.
- Status Interrupt: Select "None" to indicate that the status interrupt is not being used in the application, or select any of the 19 programmable interrupts from "*P0I*" to "*P18I*" to which this interrupt should be mapped.
- 2. Buffers:

*Case 1: FFT Points*  $(N) \leq 256$ : Following three buffers are required:

Input Buffer of size 2\*N

Output Buffer of size 2\*N

Coefficient Buffer of size 2\*N

*Case 2: FFT Points (N)*>256: For points > 256, the FFT is calculated in three steps:

Compute V Point FFT. Store this result in an intermediate output buffer.

Multiply each of the elements of this output buffer with its corresponding special twiddle coefficients.

Compute H Point FFT of the resulting output buffer. This will give the final output buffer.



Because of this, following buffers are required:

- i. Input Buffer of size 2\*N.
- ii. Output Buffer of size 2\*N.
- iii. Twiddle Coefficient Buffer for V Point FFT of size 2\*V. (Value of V for a particular value of N will be displayed in the GUI itself.)
- iv. Twiddle Coefficient Buffer for V Point FFT of size 2\*H. (Value of V for a particular value of N will be displayed in the GUI itself.)
- v. Intermediate output buffer to store the intermediate results of size 2\*N.
- vi. Special coefficient buffer of size 4\*N.
- □ Select Buffer: Select the buffer to be configured.
- Name: Enter the name of the base of the buffer.
- □ Length: The size of the buffers is fixed and not user-configurable. Thus, this field is read-only. You can view this field to see how much size to reserve for each buffer.

**NOTE:** To get the expected output, ensure that all the elements (2\*N) of the input buffer are initialized.

- □ Changing any field under Buffers enables the Save Buffer Settings button. Click this button to save the new settings. The button will be disabled automatically after it is clicked.
- 3. Under Include Code for, select the FFT check box.

#### **Code Generation**

Once all the settings are entered and the required modules are selected, perform the following steps to generate the code:

- 1. To generate the code in assembly, select the Assembly button under Language. To generate the code in C, select the c button under Language.
- 2. Click Generate Code button to save the generated file in the required project folder.

#### Using the Generated Source File

Perform the following steps to use the generated source file FIR\_IIR\_FFT\_Init.asm or FIR\_IIR\_FFT\_Init.c in a VisualDSP++ project:

- 1. Add the file FIR\_IIR\_FFT\_Init.asm or FIR\_IIR\_FFT\_Init.C to the project.
- 2. Assembly language
  - □ Declare the buffers to be used by the file FIR\_IIR\_FFT\_Init.asm as global in the main source file, for example: global FIR\_IP\_buff1;
  - Declare the subroutine to be called from FIR\_FFT\_Init.asm as external, for example: .extern \_Init\_FIR;



- □ Define and initialize all the buffers, and then call the \_Init\_FIR, \_Init\_IIR, or \_Init\_FFT function.
- 3. C language
  - □ Declare the buffers to be used by the file FIR\_IIR\_FFT\_Init.C in global memory space in the main source file, for example: global FIR\_IP\_buff1[SIZE]={ #include "input.dat" };
  - □ Declare the subroutine to be called from FIR\_FFT\_Init.C as external, for example: .extern void\_Init\_FIR();
  - □ Once all the buffers are initialized, call the Init\_FIR(), Init\_IIR(), or Init\_FFT() function.

Figure 15 shows the FIR\_IIR\_FFT\_Init.asm file for a particular configuration of the FIR accelerator.



Figure 15. "FIR\_FFT\_Init.asm" for FIR Accelerator

Figure 16 shows an example VisualDSP++ project using FIR\_IIR\_FFT.asm.





Figure 16. VisualDSP++ Project using the "FIR\_IIR\_FFT\_Init.asm" file

Figure 17 shows the FIR\_IIR\_FFT\_Init.C file for a particular configuration of the FFT accelerator.



Figure 17. FIR\_FFT\_Init.C for FFT Accelerator



Figure 18 shows an example VisualDSP++ project using FIR\_IIR\_FFT.C.

🖿 Analog Devices VisualDSP++ - [Targ	et: ADSP-21469 via HPPCI-ICE] - [Project: 64_Points] - [64_Points.c]
File Edit Session View Project Regis	ter Memory Debug Settings Tools Window Help
💌 🗅 🛩 🖃 🕼 🖏 🐇	፠ 🖻 🛍 🖾 🖂 🦓 🖓 ỡ 💷 🔺 % % 🥻 🙀 😵 💡
🔹 🛪   💁 💁 🐝 📾 🛲 🍝 -	64_Points 🗸 Debug
	9 (9 1) er (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)
Project: 64_Points.up)	* This is to test working of the GUI Generated Code for
Project Group (I project)     64 Points	#include <def21469.h></def21469.h>
🖨 🤤 Source Files	<pre>#include<cdef21469.h> #include<cigrap1 b=""></cigrap1></cdef21469.h></pre>
■ 64_Points.c	#include <stdio_h></stdio_h>
Linker Files	extern void init_FFI().
🔲 Header Files	void FFT_DMA_ISR();
	#define N 64
	<pre>float FFT_IP_buff[2*N]={</pre>
	#include "input.dat"
	}:
	float FFT_OP_buff[2*N];
	<pre>float FFT_CF_buff[2*N]={</pre>
	}: #include "twiddle_64.dat"
	int count:
	int main()
	int x;
	interrupt(SIG P0.FFT DMA ISR);
	Init EFT():
	x=U; }

Figure 18. VisualDSP++ project using the "FIR\_IIR\_FFT\_Init.C"

## **Example Code and Other Files**

In addition to the Expert Code Generator, a number of assembly and C example projects are provided in the associated .zip file. Along with these code examples, all the twiddle coefficients needed for the FFT accelerator are supplied in .dat files. Those are:

- twiddle\_16.dat: Twiddle coefficients for 16 points FFT (buffer size = 32)
- twiddle\_32.dat: Twiddle coefficients for 32 points FFT (buffer size = 64)
- twiddle\_64.dat: Twiddle coefficients for 64 points FFT (buffer size = 128)
- twiddle\_128.dat: Twiddle coefficients for 128 points FFT (buffer size = 256)
- twiddle\_256.dat: Twiddle coefficients for 256 points FFT (buffer size = 512)

Special twiddle coefficients:

- pw\_twiddle\_512.dat: Special twiddle coefficients for 512 points FFT (buffer size = 2048)
- pw\_twiddle\_1024.dat: Special twiddle coefficients for 1024 points FFT (buffer size = 4096)
- pw\_twiddle\_2048.dat: Special twiddle coefficients for 2048 points FFT (buffer size = 8192)
- pw\_twiddle\_4096.dat: Special twiddle coefficients for 4096 points FFT (buffer size = 16384)
- pw\_twiddle\_8192.dat: Special twiddle coefficients for 8192 points FFT (buffer size = 32768). s



### References

- [1] Managing the Core PLL on SHARC Processors (EE-290). Rev 4, November. Analog Devices, Inc.
- [2] ADSP-2146x SHARC Processor Hardware Reference. Rev 0.3, July 2010. Analog Devices, Inc.
- [3] ADSP-21261/ADSP-21262/ADSP-21266 SHARC Processor Data Sheet. Rev F, August 2009. Analog Devices, Inc.
- [4] ADSP-21362/ADSP-21363/ADSP-21364/ADSP-21365/ADSP-21366 SHARC Processor data sheet. Rev G, March 2011. Analog Devices, Inc.
- [5] ADSP-21367/ADSP-21368/ADSP-21369 SHARC Processors Data Sheet. Rev E, July 2009. Analog Devices, Inc.
- [6] ADSP-21371/ADSP-21375 SHARC Processor Data Sheet. Rev C, September 2009. Analog Devices, Inc.
- [7] ADSP-21467/ADSP-21469 SHARC Processor Data Sheet. Rev A, December, 2011. Analog Devices, Inc.
- [8] ADSP-21478/ADSP-21479 SHARC Processor Data Sheet. Rev A, April 2011. Analog Devices, Inc.
- [9] ADSP-21483/21486/21487/21488/21489 SHARC Processor Data Sheet. Rev 0, January 2011. Analog Devices, Inc.
- [10] ADSP-21467/ADSP-21469 SHARC Anomaly List for Revisions 0.0. Rev F, November 2011. Analog Devices, Inc.
- [11] ADSP-21478/ADSP-21479 SHARC Anomaly List for Revisions 0.0, 0.1, 0.2. Rev C, September 2011. Analog Devices, Inc.
- [12] ADSP-21483/ADSP-21486/ADSP21487/ADSP-21488/ADSP-21489 SHARC Anomaly List for Revisions 0.1, 0.2. Rev F, September 2011. Analog Devices, Inc.
- [13] MT48LC4M32B2 Data Sheet. Rev E, October, 2002. Micron Technology, Inc.
- [14] MT48LC8M16A2 Data Sheet. Rev N, January 2009. Micron Technology, Inc.
- [15] MT48LC16M16A2 Data Sheet. Rev N, January 2010. Micron Technology, Inc.
- [16] MT47H64M16 Data Sheet. Rev P, January 2009, Micron Technology, Inc.



## **Document History**

Revision	Description
Rev 5 – January 17, 2012 by Mitesh Moonat	Updated the plug-in to add fix for the ADSP-214xx PLL anomaly 15000020. Added minor fixes in "cs_macros.h" file. Verified the plug-in and example code functionality under VisualDSP++ 5.0 Update 10.
Rev 4 – July 13, 2010 by Mitesh Moonat	Added support for ADSP-2126x, ADSP-213xx, ADSP-2147x, and ADSP-2148x SHARC processors. Renamed "cs2146x.h" to "cs_macros.h". Verified the plug-in and example code functionality under VisualDSP++ 5.0 Update 8.
Rev 3 – December 17, 2009 by Mitesh Moonat	Updated the header file "cs2146x.h" and modified the plug-ins to address code generation issues. Also, verified plug-ins and example code functionality under VisualDSP++ 5.0 Update 7.
	Furthermore, added information on registering the plug-ins under the Windows Vista operating system.
Rev 2 – September 24, 2009 by Mitesh Moonat	Added Save/Load/Reset options; corrected DDR2CLK calculation, default DDR2 controller settings, and MLBCLK:CCLK supported ratios. Also, updated the def21469.h, Cdef21469.h, and cs2146x.h files. In addition, the updated plug-in takes now care of the extended PMCTL register's effect latency.
Rev 1 – May 29, 2009 by Mitesh Moonat	Initial release.