Spring 2023

EE 382N-4: Advanced Embedded Systems

Lab Assignment #1

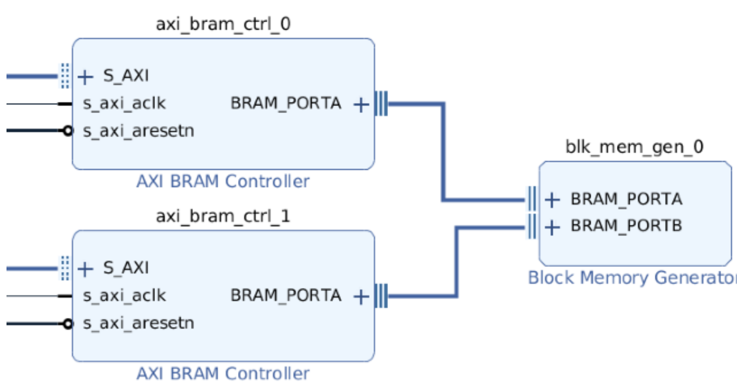
DUE FEB 2ND, 2023

Lab Goals:

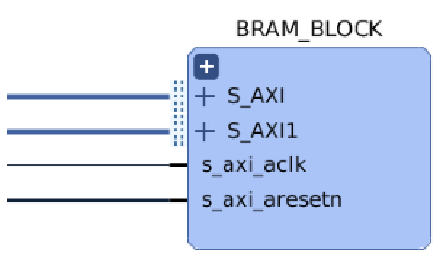
This lab introduces the student to the Xilinx Vivado Design Environment and programming in the Linux environment. The student will learn how to use the IP building blocks in the Zynq FPGA system and how to build a custom IP block using an AXI Peripheral template. The circuit implements a dual-ported memory that will be used to generate a memory test environment. The memory test will be performed while dynamically changing the processor clock speed and the Programmable Logic (PL) clock speed.

Schematic design procedure:

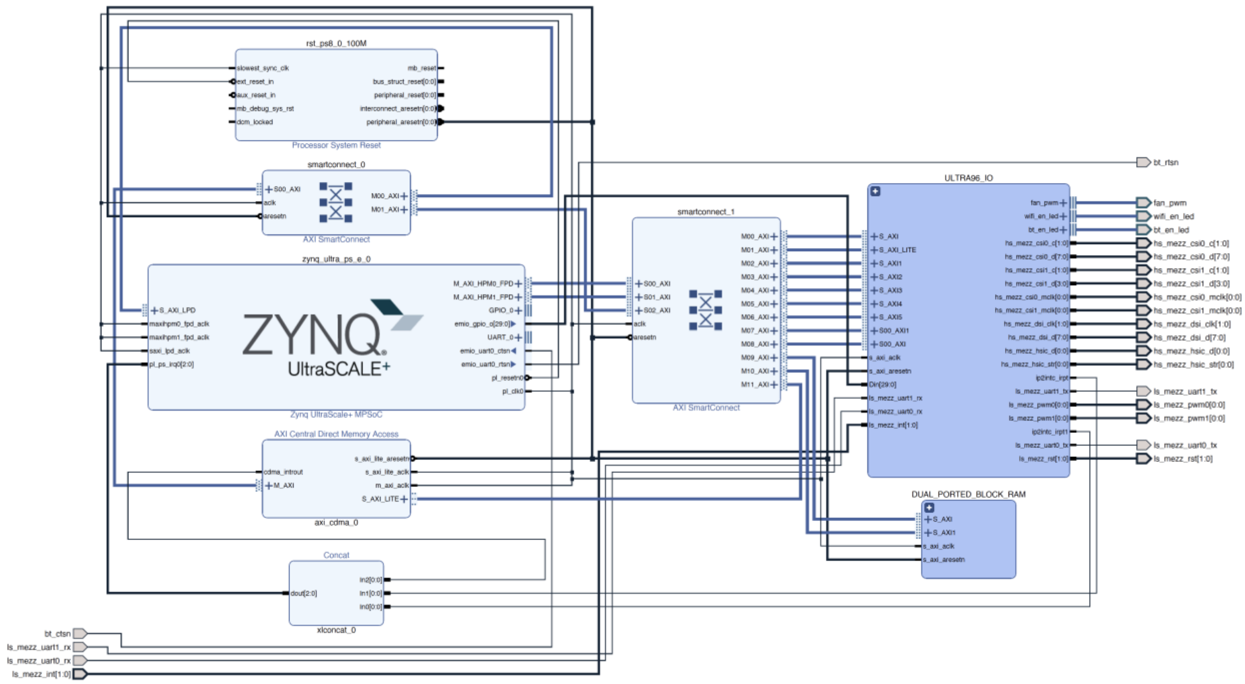
1) Refer to the following document to set up your Vivado environment
2) Using the BASELINE_ULTRA96 schematic instantiate a "True Dual Port" 2048x32 (8K bytes) BRAM and two BRAM Controllers as shown below:



3) Select all 3 symbols and create a level of hierarchy:

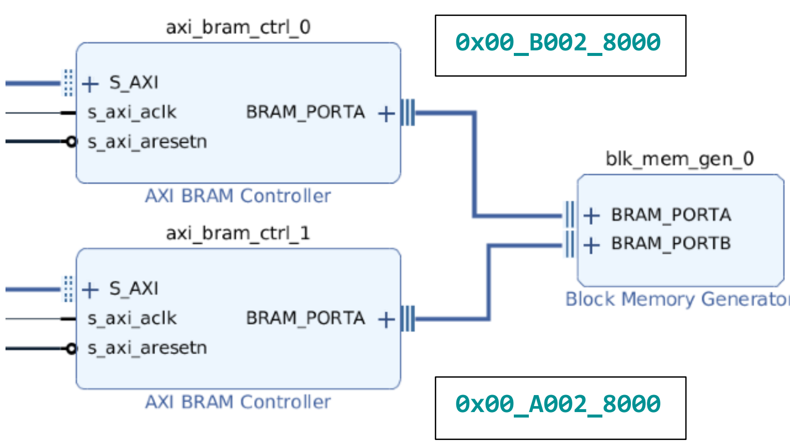


The final schematic should look something like this after running the Block Connection Automation routine:



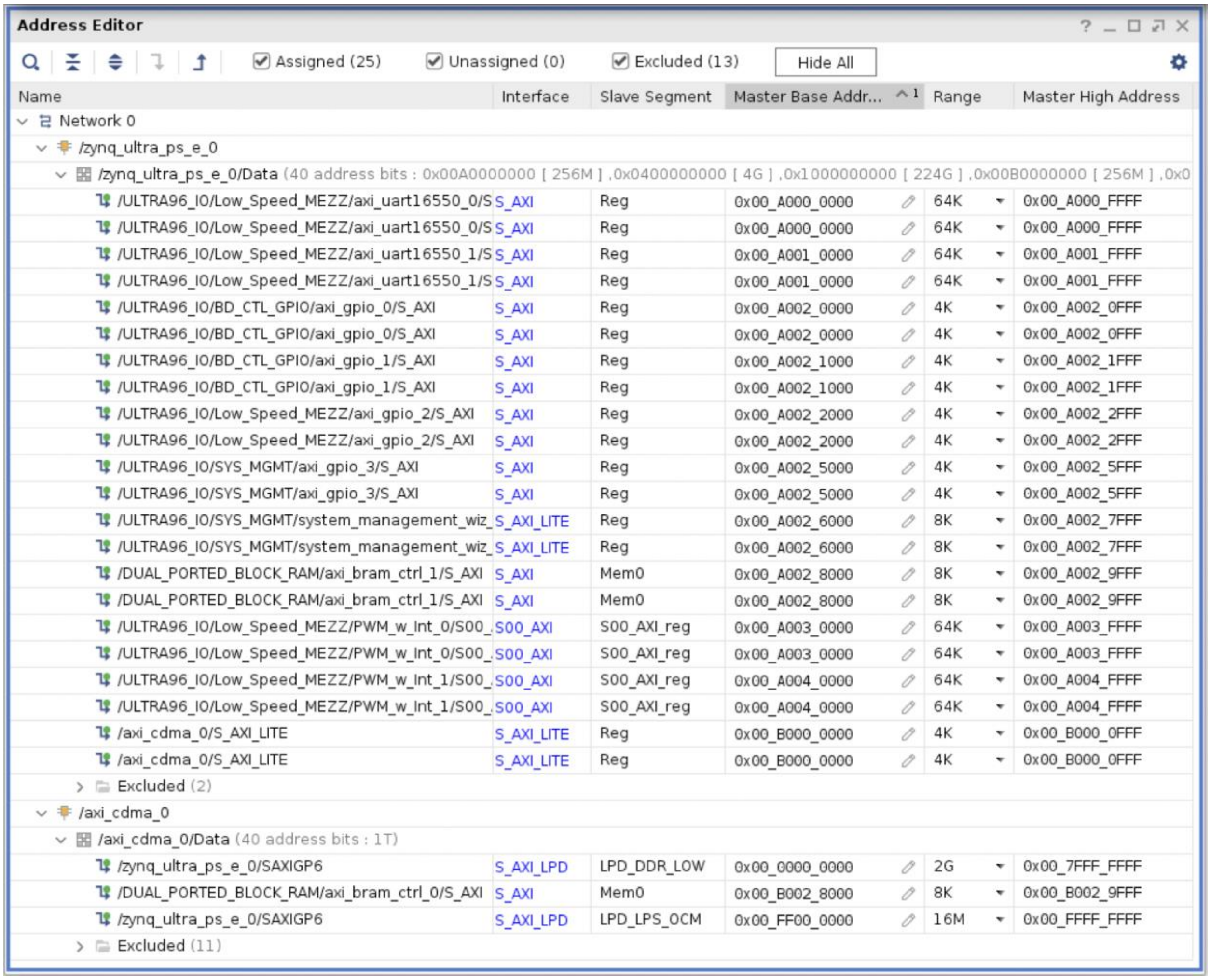
NOTE: Vivado will redraw the schematic <u>differently</u> every time you make a change.

You will need to assign addresses to new BRAM controllers:



It is critical that you use the addresses shown above for all the components in the schematic. The Device Tree Blob (DTB) uses these addresses to tell the Linux Kernel what devices are active and what address the devices are located at.

The resulting address map should look like this:



Name	Interface	Slave Segment	Master Base Addr...	Range	Master High Address
Network 0					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x0					
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_0/S	S_AXI	Reg	0x00_A000_0000	64K	0x00_A000_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_0/S	S_AXI	Reg	0x00_A000_0000	64K	0x00_A000_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_1/S	S_AXI	Reg	0x00_A001_0000	64K	0x00_A001_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_1/S	S_AXI	Reg	0x00_A001_0000	64K	0x00_A001_FFFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A002_0000	4K	0x00_A002_0FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A002_0000	4K	0x00_A002_0FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_1/S_AXI	S_AXI	Reg	0x00_A002_1000	4K	0x00_A002_1FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_1/S_AXI	S_AXI	Reg	0x00_A002_1000	4K	0x00_A002_1FFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_gpio_2/S_AXI	S_AXI	Reg	0x00_A002_2000	4K	0x00_A002_2FFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_gpio_2/S_AXI	S_AXI	Reg	0x00_A002_2000	4K	0x00_A002_2FFF
/ULTRA96_IO/SYS_MGMT/axi_gpio_3/S_AXI	S_AXI	Reg	0x00_A002_5000	4K	0x00_A002_5FFF
/ULTRA96_IO/SYS_MGMT/axi_gpio_3/S_AXI	S_AXI	Reg	0x00_A002_5000	4K	0x00_A002_5FFF
/ULTRA96_IO/SYS_MGMT/system_management_wiz_	S_AXI_LITE	Reg	0x00_A002_6000	8K	0x00_A002_7FFF
/ULTRA96_IO/SYS_MGMT/system_management_wiz_	S_AXI_LITE	Reg	0x00_A002_6000	8K	0x00_A002_7FFF
/DUAL_PORTED_BLOCK_RAM/axi_bram_ctrl_1/S_AXI	S_AXI	Mem0	0x00_A002_8000	8K	0x00_A002_9FFF
/DUAL_PORTED_BLOCK_RAM/axi_bram_ctrl_1/S_AXI	S_AXI	Mem0	0x00_A002_8000	8K	0x00_A002_9FFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_0/S00_	S00_AXI	S00_AXI_reg	0x00_A003_0000	64K	0x00_A003_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_0/S00_	S00_AXI	S00_AXI_reg	0x00_A003_0000	64K	0x00_A003_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_1/S00_	S00_AXI	S00_AXI_reg	0x00_A004_0000	64K	0x00_A004_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_1/S00_	S00_AXI	S00_AXI_reg	0x00_A004_0000	64K	0x00_A004_FFFF
/axi_cdma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_B000_0000	4K	0x00_B000_0FFF
/axi_cdma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_B000_0000	4K	0x00_B000_0FFF
Excluded (2)					
/axi_cdma_0					
/axi_cdma_0/Data (40 address bits : 1T)					
/zynq_ultra_ps_e_0/SAXIGP6	S_AXI_LPD	LPD_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
/DUAL_PORTED_BLOCK_RAM/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0x00_B002_8000	8K	0x00_B002_9FFF
/zynq_ultra_ps_e_0/SAXIGP6	S_AXI_LPD	LPD_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
Excluded (11)					

The ***system.dtb*** file that corresponds to the address map above can be downloaded from here: http://projects.ece.utexas.edu/courses/spring_23/ee382n4-17685/arch/labs/SP23_LAB_1/

The next step is to generate a bit file that can be dynamically downloaded into the FPGA. Refer to this guide on how to generate the FPGA bit file: BIT File Generation Flow

Lab procedure:

In this Lab you will need to use the Linux **`srand(time(0))`** and **`rand()`** routines to generate random data and addresses for the tests. For a code example see the **`frm`** command. NOTE: a new random seed must be generated each time a new test is invoked. In addition to randomly varying the address and data you will need to be randomly varying the clock frequencies of the CPU and PL. The frequencies that you need to use are shown to the right. There are 25 total combinations. All combinations must be programmed for Test #1 and Test #3. For test #2 use all 5 PS clock frequencies.

PS (CPU) Clock	**PL (FPGA) Clock**
1499 MHz	300 MHz
1250 MHz	266 MHz
1000 MHz	187.5 MHz
858 MHz	150 MHz
416.6 MHz	100 MHz

Test #1: Write a BRAM (@0xA002_8000) memory (2 pages -- 8K bytes) test that randomly varies the address and data while randomly varying the CPU and PL clock frequencies as shown in the table above. The test should run continuously until interrupted with a control-c.

Test #2: Write a memory test that does a pseudo random address and data test of the OCM memory (128K bytes) while varying the CPU clock frequency only. The test should run continuously until interrupted with a control-c.

Test #3: The memory will be tested using a four-step process:

1. Load a memory page (i.e., 4K bytes) in the OCM with a 1024 random 32-bit data values using the Linux **`srand(time(0))`** and **`rand()`** routines. Use 0xFFFC_0000 as the starting address of this page.
2. Transfer the random data in the OCM (0xFFFC_0000) to the BRAM (0xB002_8000) using the CDMA unit.
3. Transfer the random data in the BRAM (0xB002_8000) back to the OCM at a different address (0xFFFC_2000) using the CDMA unit.
4. Once the 3 steps above are complete, the OCM data at address 0xFFFC_0000 is compared to the OCM data at address 0xFFFC_2000 using SHA-256 hashing. The procedure is to run the hash on both pages (0xFFFC_0000 & 0xFFFC_2000) and then compare results. The test should run continuously until interrupted with a control-c.
5. This confirms that DMA traffic to/from the OCM & BRAM works. Here is some example CODE

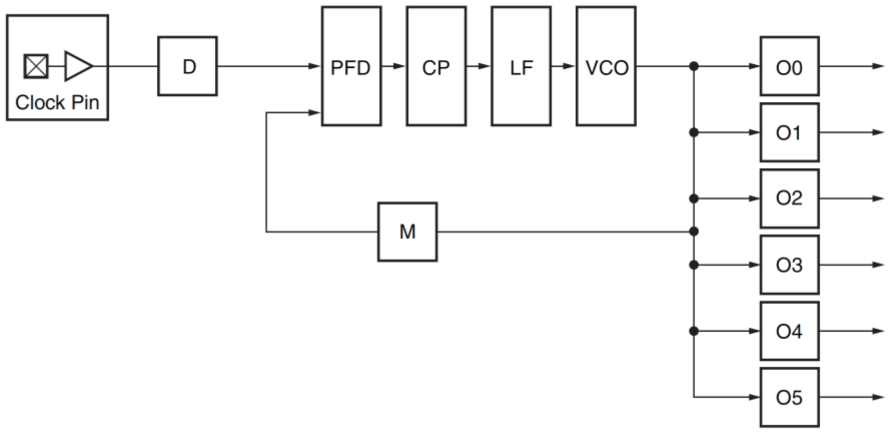
The memory tests will be written in C and compiled on the Ultra96 using the GNU tool suite. The user input to the test must accept the following inputs:

1. Number of test loops -- Default: continuous
2. Number of 32-bit words to be tested -- Default: 2048 (BRAM), 4096 (OCM)
3. The memory test will display the following output upon successful completion of the test: Test passed: "xx" Loops of "yy" 32-bit words

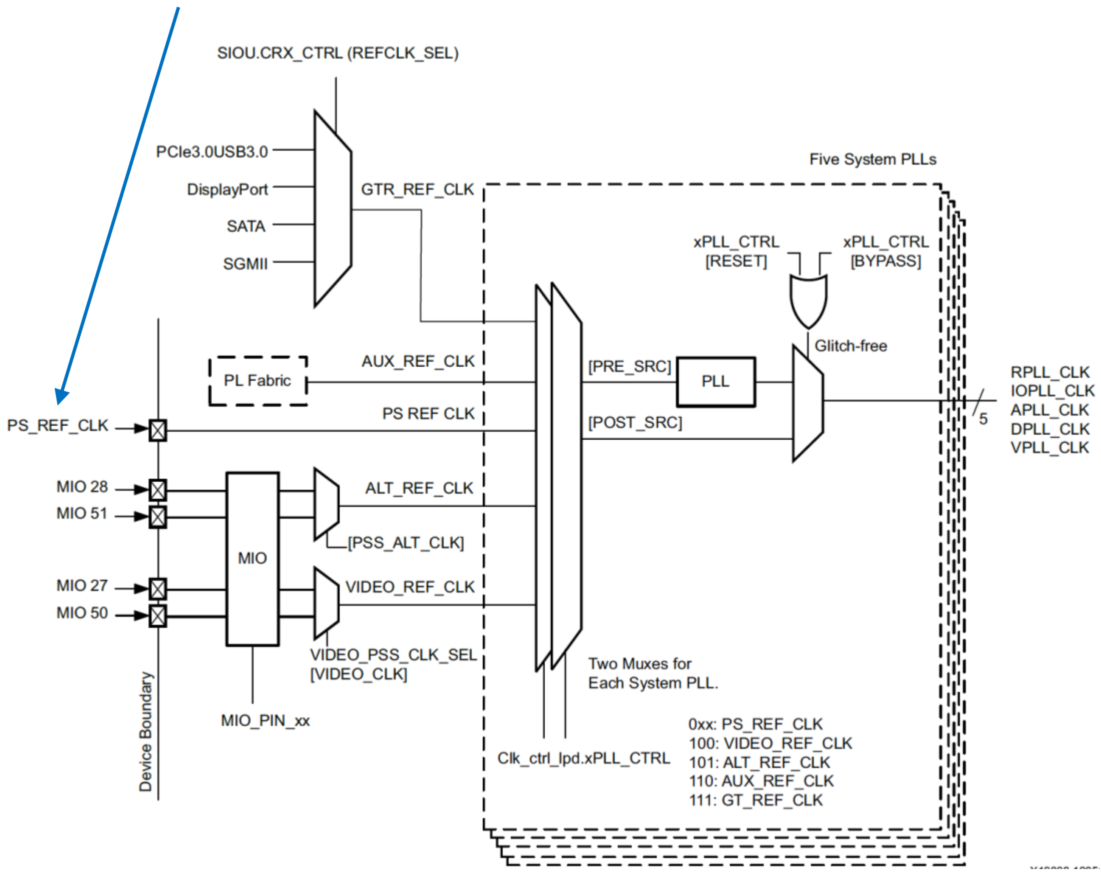
Do NOT use the Vivado SDK development tools. They are for bare-metal implementations. We will not be doing any bare-metal implementations in this class.

Background Information

The figure below shows a block diagram of a typical PLL in the ZynqMP SOC. The “M” block is the feedback divider (FBDIV) which divides the output of the VCO and feeds the divided clock into the Phase-Frequency-detector (PFD) providing the frequency multiplication factor of a PLL.



The figure below shows the block diagram of the 5 system PLLs in the ZynqMP. The APLL provides clocks to the Processing System (PS). The IOPLL provides a clock to the Programmable Logic (PL). The Ultra96 PS_REF_CLK is a 33.3333MHz clock.



The following web page specifies all the accessible registers on the ZynqMP SOC:
https://www.xilinx.com/html_docs/registers/ug1087/ug1087-zynq-ultrascale-registers.html

There are three registers that affect the PS clock frequency. An example of how to set the APLL frequency is shown in Appendix A.

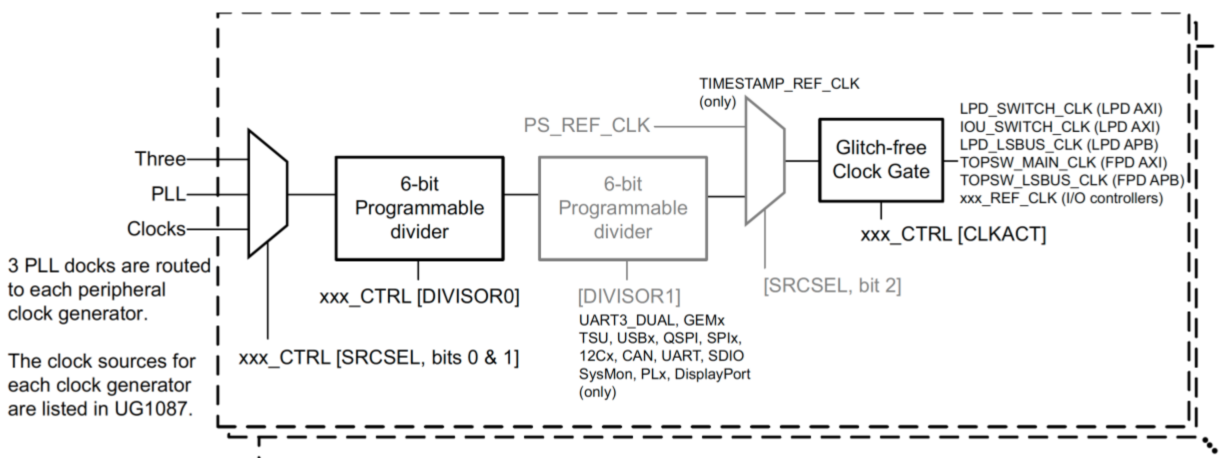
Register Name	APLL_CTRL	This register controls the FBDIV and CLKOUT values of the APLL
Absolute Address	0x00FD1A0020 (CRF_APB)	

Register Name	APLL_CFG	This register controls how quickly the PLL locks
Absolute Address	0x00FD1A0024 (CRF_APB)	

Register Name	PLL_STATUS	This register is the "LOCK" status of the APLL when changing frequencies.
Absolute Address	0x00FD1A0044 (CRF_APB)	

There is one register that affect the PL clock frequency:

Register Name	PL0_REF_CTRL	This register controls the two 6-bit clock dividers (see block diagram below)
Absolute Address	0x00FF5E00C0 (CRL_APB)	



Refer to Appendix B for an example on how to change the PL clock frequency.

Appendix A: PS Integer Multiply and Divide Programming Example

The Ultra96 FBDIV reset value is 0x48 (72) and the Output Divider is set to divide by 2. The PS clock frequency is: 33.3333MHz * 72 / 2 = 1199 MHz

Let's set the PS clock frequency to 1499 MHz. For a new frequency of 1499 MHz, the [FBDIV] value is switched to 45 (0x2D) and the output divider is set to 0x0.

NOTE: Before reprogramming the PLL clock output frequency, check that the downstream clocks are in a safe state before releasing. For example, the APU DIVISOR must be set to 2.

1. Program the new FBDIV, CLKOUT value (do NOT modify other values in the APLL_CTRL register):
   Set APLL_CTRL = 0x0000_2D00: [DIV2] = 0x0, [FBDIV] = 0X2D
   `pm 0xfd1a0020 0x00002D00`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
--	--	--	--	--	POST SRC			--	PRE SRC			--	--	--	DIV2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
--	FBDIV							--	--	--	--	BYPASS	--	--	RST
0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0

2. Program the control data for APLL_CFG using the data in Table 1.

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
45	3	12	3	63	825

Set APLL_CFG[31:25] = 0x3F // LOCK_DLY
Set APLL_CFG[22:13] = 0x339 // LOCK_CNT
Set APLL_CFG[11:10] = 0x3 // LFHF
Set APLL_CFG[8:5] = 0x3 // CP
Set APLL_CFG[3:0] = 0x12 // RES

31	30	29	28	27	26	25	24	23
LOCK_DLY							--	--
0	1	1	1	1	1	1	0	0

22	21	20	19	18	17	16	15	14	13
LOCK_CNT									
1	1	0	0	1	1	1	0	0	1

12	11	10	9	8	7	6	5	4	3	2	1	0
--	LFHF		--	CP				--	RES			
0	1	1	0	0	0	1	1	0	1	1	0	0

Set APLL_CFG[31:0] = 0x7E67_2C6C
`pm 0xfd1a0024 0x7E672C6C`

3. Program the bypass:
   Set APLL_CTRL[31:0] = 0x0000_2D08h: [BYPASS] = 0x1
   `pm 0xfd1a0020 0x00002D08`
4. Assert reset. This is when the new data is captured into the PLL.
   Set APLL_CTRL[31:0] = 0x0000_2D09h: [BYPASS] = 0x1 & [RESET] = 0x1
   `pm 0xfd1a0020 0x00002D09`
5. Deassert reset.
   Set APLL_CTRL[31:0] = 0x0000_2D08h: [BYPASS] = 0x1 [RESET] = 0x0
   `pm 0xfd1a0020 0x00002D08`
6. Check for LOCK. Wait until: PLL_STATUS [APLL_LOCK] = 0x1
   `while (dm 0xfd1a0044 != 0x1) do wait   // Pseudo code does NOT work in the CLI`
7. Deassert bypass.
   Set APLL_CTRL[31:0] = 0x0000_2D00h: [BYPASS] = 0x00
   `pm 0xfd1a0020 0x00002D00`

The PLL output clock is now set to 1499 MHz.

Table 1: PLL Integer Feedback Divider Helper Data Values

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
25	3	10	3	63	1000
26	3	10	3	63	1000
27	4	6	3	63	1000
28	4	6	3	63	1000
29	4	6	3	63	1000
30	4	6	3	63	1000
31	6	1	3	63	1000
32	6	1	3	63	1000
33	4	10	3	63	1000
34	5	6	3	63	1000
35	5	6	3	63	1000
36	5	6	3	63	1000
37	5	6	3	63	1000
38	5	6	3	63	975
39	3	12	3	63	950
40	3	12	3	63	925
41	3	12	3	63	900
42	3	12	3	63	875
43	3	12	3	63	850

44	3	12	3	63	850
45	3	12	3	63	825
46	3	12	3	63	800
47	3	12	3	63	775
48	3	12	3	63	775
49	3	12	3	63	750
50	3	12	3	63	750
51	3	2	3	63	725
52	3	2	3	63	700
53	3	2	3	63	700
54	3	2	3	63	675
55	3	2	3	63	675
56	3	2	3	63	650
57	3	2	3	63	650
58	3	2	3	63	625
59	3	2	3	63	625
60	3	2	3	63	625
61 to 82	3	2	3	63	600
83 to 102	4	2	3	63	600
103	5	2	3	63	600
104	5	2	3	63	600
105	5	2	3	63	600
106	5	2	3	63	600
107 to 125	3	4	3	63	600

Appendix B: PL Clock Control code example

```
#include "stdio.h"
#include "stdlib.h"
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char * argv[])
{
  int dh = open("/dev/mem", O_RDWR | O_SYNC);
  if(dh == -1) {
        printf("Unable to open /dev/mem\n");
  }

uint32_t* clk_reg = mmap(NULL,
                         0x1000,
                         PROT_READ|PROT_WRITE,
                         MAP_SHARED, dh, 0xFF5E0000);
int i = 0;
uint32_t* pl0 = clk_reg;

pl0+=0xC0;                              // PL0_REF_CTRL reg offset 0xC0

*pl0 = (1<<24)                          // bit 24 enables clock
     | (1<<16)                          // bit 23:16 is divisor 1
     | (6<<8);                          // bit 15:0 is clock divisor 0
                                        // frequency = 1.5Ghz/divisor0/divisor1
                                        //           = 1.5Ghz/6=250MHz

  munmap(clk_reg, 0x1000);
  return 0;
}
```

Field Name	Bits	Type	Reset Value	Description
Reserved	31:25	rw	0x0	reserved
CLKACT	24	rw	0x0	Clock active control. 0: disable. Clock stop. 1: enable.
Reserved	23:22	rw	0x0	reserved
DIVISOR1	21:16	rw	0x5	6-bit divider.
Reserved	15:14	rw	0x0	reserved
DIVISOR0	13:8	rw	0x20	6-bit divider.
Reserved	7:3	rw	0x0	reserved
SRCSEL	2:0	rw	0x0	Clock generator input source. 000: IOPLL 010: RPLL 011: DPLL_CLK_TO_LPD

Appendix C: PS Clock Control Registers

APLL_CTRL (CRF_APB) Register Description

Register Name	APLL_CTRL
Relative Address	0x0000000020
Absolute Address	0x00FD1A0020 (CRF_APB)
Width	32
Type	rw
Reset Value	0x00012C09
Description	APLL Clock Unit Control

APLL_CTRL (CRF_APB) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
POST_SRC	26:24	rw	0x0	Select the pass-thru clock source for PLL Bypass mode. 0xx: PS_REF_CLK 100: VIDEO_REF_CLK 101: ALT_REF_CLK 110: AUX_REF_CLK 111: GT_REF_CLK
PRE_SRC	22:20	rw	0x0	Select the clock source for PLL input. 0xx: PS_REF_CLK 100: VIDEO_REF_CLK 101: ALT_REF_CLK 110: AUX_REF_CLK 111: GT_REF_CLK
DIV2	16	rw	0x1	Enable the divide by 2 function inside of the PLL. 0: no effect. 1: divide clock by 2. Note: this does not change the VCO frequency, just the output frequency.
FBDIV	14:8	rw	0x2C	Feedback divisor integer portion for the PLL.
BYPASS	3	rw	0x1	PLL Clock Bypass Mode. 0: normal PLL mode; the source clock is selected using [PRE_SRC]. 1: bypass the PLL; the source clock is selected using [POST_SRC].
RESET	0	rw	0x1	PLL reset. 0: active. 1: reset. Note: Program the PLL into bypass mode before resetting the PLL.

APLL_CFG (CRF_APB) Register Description

Register Name	APLL_CFG
Relative Address	0x0000000024
Absolute Address	0x00FD1A0024 (CRF_APB)
Width	32
Type	rw
Reset Value	0x00000000
Description	APLL Integer Helper Data Configuration.

APLL_CFG (CRF_APB) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
LOCK_DLY	31:25	rw	0x0	Lock circuit configuration settings for lock windowsize
LOCK_CNT	22:13	rw	0x0	Lock circuit counter setting
LFHF	11:10	rw	0x0	PLL loop filter high frequency capacitor control
CP	8:5	rw	0x0	PLL charge pump control
RES	3:0	rw	0x0	PLL loop filter resistor control

PLL_STATUS (CRF_APB) Register Description

Register Name	PLL_STATUS
Relative Address	0x0000000044
Absolute Address	0x00FD1A0044 (CRF_APB)
Width	8
Type	mixed
Reset Value	0x00000038
Description	FPD PLL Clocking Status.

PLL_STATUS (CRF_APB) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
VPLL_STABLE	5	ro	0x1	VPLL stability status. 0: not locked or bypassed. 1: locked or bypassed.
DPLL_STABLE	4	ro	0x1	DPLL stability status. 0: not locked or bypassed. 1: locked or bypassed.
APLL_STABLE	3	ro	0x1	APLL stability status. 0: not locked or bypassed. 1: locked or bypassed.
VPLL_LOCK	2	ro	0x0	VPLL lock status. 0: not locked. 1: locked.
DPLL_LOCK	1	ro	0x0	DPLL lock status. 0: not locked. 1: locked.
APLL_LOCK	0	ro	0x0	APLL lock status. 0: not locked. 1: locked.

Appendix D: PL Clock Control Registers

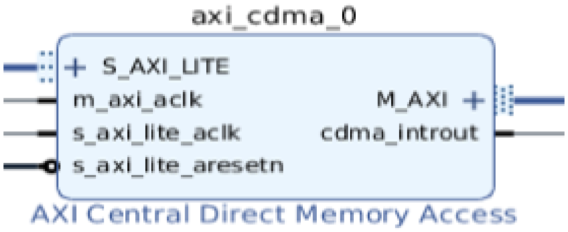
PL0_REF_CTRL (CRL_APB) Register Description

Register Name	PL0_REF_CTRL
Relative Address	0x00000000C0
Absolute Address	0x00FF5E00C0 (CRL_APB)
Width	32
Type	rw
Reset Value	0x00052000
Description	PL 0 Clock Generator Config.

PL0_REF_CTRL (CRL_APB) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
Reserved	31:25	rw	0x0	reserved
CLKACT	24	rw	0x0	Clock active control. 0: disable. Clock stop. 1: enable.
Reserved	23:22	rw	0x0	reserved
DIVISOR1	21:16	rw	0x5	6-bit divider.
Reserved	15:14	rw	0x0	reserved
DIVISOR0	13:8	rw	0x20	6-bit divider.
Reserved	7:3	rw	0x0	reserved
SRCSEL	2:0	rw	0x0	Clock generator input source. 000: IOPLL 010: RPLL 011: DPLL_CLK_TO_LPD

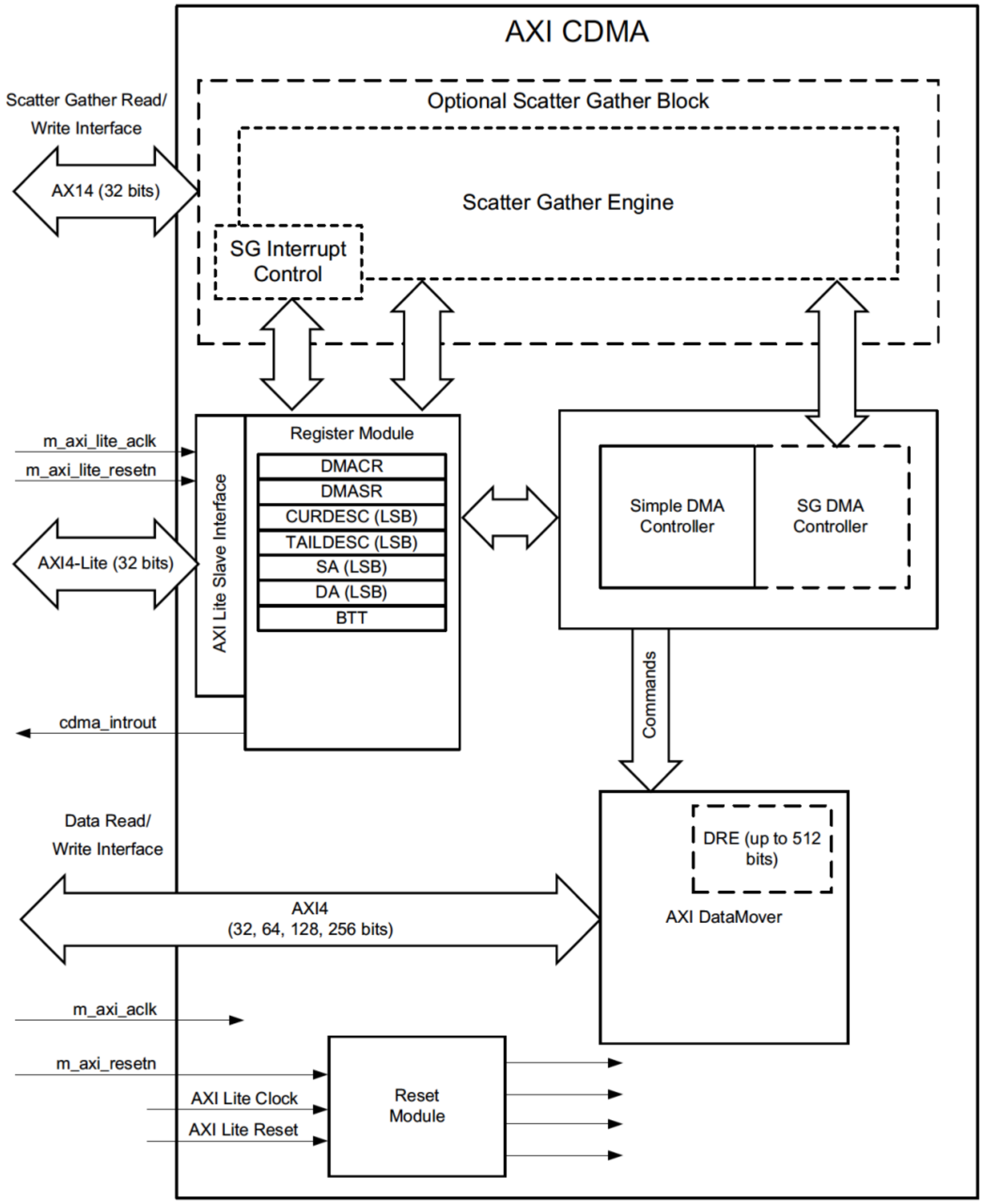
Appendix E: CDMA details



https://www.xilinx.com/products/intellectual-property/axi_central_dma.html

Signal Name	**Interface**	**Signal Type**	**Init Status**	**Description**
		System Signals		
m_axi_aclk	Clock	I	–	AXI CDMA Synchronization Clock
cdma_introut	Interrupt	O	0	Interrupt output for the AXI CDMA core
		AXI4-Lite Slave Interface Signals		
s_axi_lite_aclk	S_AXI_LITE	I	–	Synchronization clock for the AXI4-Lite interface. This clock can be the same as m_axi_aclk (synchronous mode) or different (asynchronous mode). ***Note:*** If it is asynchronous, the frequency of this clock must be less than or equal to the frequency of the m_axi_aclk.
s_axi_lite_aresetn	S_AXI_LITE	I	–	Active-Low AXI4-Lite Reset. When asserted Low, the AXI4-Lite Register interface and the entire CDMA core logic is put into hard reset. This signal must be synchronous to s_axi_lite_aclk.
s_axi_lite_*	S_AXI_LITE	I/O	–	See Appendix A of the *Vivado AXI Reference Guide* (UG1037)[Ref 2] for AXI4 signal.
		CDMA Data AXI4 Read/Write Master Interface Signals		
m_axi_*	M_AXI	I/O	–	See Appendix A of the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for AXI4 signal.
		Scatter Gather AXI4 Read/Write Master Interface Signals		
m_axi_sg_*	M_AXI_SG	I/O	–	See Appendix A of the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for AXI4 signal.

C_M_AXI_BURST_LEN	**Test Packet Size**	**AXI4 Frequency**	**Observed Bus Bandwidth Utilization by AXI CDMA**
16	9,000 bytes	150 MHz	70%
64	9,000 bytes	150 MHz	up to 99%



AXI CDMA
Optional Scatter Gather Block
Scatter Gather Engine
SG Interrupt Control
Scatter Gather Read/ Write Interface
AX14 (32 bits)
m_axi_lite_aclk
m_axi_lite_resetn
AXI4-Lite (32 bits)
AXI Lite Slave Interface
Register Module
DMACR
DMASR
CURDESC (LSB)
TAILDESC (LSB)
SA (LSB)
DA (LSB)
BTT
Simple DMA Controller
SG DMA Controller
Commands
cdma_introut
Data Read/ Write Interface
AXI4
(32, 64, 128, 256 bits)
DRE (up to 512 bits)
AXI DataMover
m_axi_aclk
m_axi_resetn
AXI Lite Clock
AXI Lite Reset
Reset Module

Xilinx Zynq UltraScale+ Tutorials and Documentation

ZYNQ UltraScale+ White Papers

ZYNQ UltraScale+ Register Map

ZYNQ UltraScale+ MPSoC Base Targeted Reference Design

ZYNQ UltaScale+ Documentation

ZYNQ UltraScale+ Video Tutorials

Zynq UltraScale+ All Programmable SoC Technical Reference Manual

Exploring Zynq MPSoC

FPGAs for SW Programmers

AXI Infrastructure Intellectual Property

Creating an AXI Peripheral

Using Xilinx SDK

Repository of useful Vivado, Zynq & Petalinux Documentation

Vivado Tutorials and Documentation

Vivado Video Tutorials

Vivado Design Suite User Guide: Getting Started (UG910)

Vivado Design Suite Tutorial (UG940)

Vivado Design User Guide: Design Flows Overview (UG892)

Vivado Design Suite Tutorial: Design Flows Overview (UG888)

Vivado Design Suite Tutorial: Programming and Debugging (UG936)

Vivado Design Suite User Guide: High-Level Synthesis (UG902)

Vivado Design Suite User Guide: Synthesis (UG901)

Vivado Design Suite User Guide: Implementation (UG904)

Introduction to FPGA Design with Vivado High-Level Synthesis

Vivado Design Suite User Guide: Using Tcl Scripting (UG894)

Vivado Design Suite Tcl Command Reference Guide (UG835)

Vivado Design Suite User Guide: Designing with IP (UG896)

Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994)

Vivado Design Suite User Guide: Logic Simulation (UG900)

Vivado Design Suite User Guide: Using Constraints (UG903)

Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)

Vivado Design Suite User Guide: Design Analysis and Closure Techniques (Design HUB)

Vivado Design Suite User Guide: Programming and Debugging (UG908)

Vivado Design Suite User Guide: System-Level Design Entry (UG895)

Vivado Design Suite Properties Reference Guide (UG912)

Vivado Design Suite User Guide: I/O and Clock Planning (UG899)

Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator (UG897)

Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)