FPGA Reliability Project

# Introduction

This project is intended to produce an online runtime floorplanning algorithm for FPGAs which minimizes hotspot generation, thermal cycling, and evenly distributes aging across the FPGA die. The other focus of the algorithm is to scrub and repair radiation-induced upsets of configuration memory in the FPGA. The configuration memory of the FPGA is defined in SRAM, which is uniquely susceptible to high-energy strikes and other radiation related events.

This project therefore has essentially three primary components. The Microblaze soft processor, the C code running on the Microblaze which performs the floorplanning, and the VHDL/Verilog code which comprises the test blocks running in the FPGA.

The Microblaze soft processor is automatically generated by the Xilinx toolchain, which will be covered later in this document. The Microblaze processor is intended to provide a monolithic solution for designers, all they need to do is integrate a FPGA with the appropriate amount of resources, and the system has both a reconfigurable digital portion, and a highly configurable (though static once programmed) processor. In this application, the processor will control the internal reconfiguration port and VHDL/Verilog peripherals to floorplan and/or simulate running a floorplan.

The C code comprises the instructions that perform the floorplanning and housekeeping activities of the FPGA. The Microblaze (as was configured in the toolchain) has connections to all the relevant peripherals to this project, and the communication channels are handled primarily over a memory mapping interface. Thus, as will be discussed later in the document, the system performs the majority of the IO with this memory mapped interface over “#define”’s.

The final portion of the project is the VHDL/Verilog modules. The modules are intended to be utilized as partially reconfigurable peripherals and are thus as modular as possible. Partially reconfigurable peripherals will be covered later in the document.

# Quick Start

The first step of installing the project is to install the Xilinx toolchain. The toolchain is available for download from the [Xilinx website](http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html) (ISE Design Suite). This project is implemented in the ISE toolchain, but Xilinx has moved to its “sustaining phase” and is “no longer recommended” for future designs. Instead, the Vivado Design Suite is recommended for new projects.

Once the rather large download (6 GB) has completed, begin the installation with the following steps.

1. Extract the Xilinx\_ISE\_DS\_Win\_14.7\_1015\_1 to the chosen root directory.
2. Execute xsetup.exe.
3. Accept all the license agreements, terms and conditions.
4. On the **Select Products to Install** page, choose the **ISE Design Suite Embedded Edition** option.
5. Select all the options on the **Select Installation Options**.
6. Use the default install path (C:\Xilinx), associate tool preferences and files by clicking the **Tool preferences and file association** option.

Once the Xilinx toolchain has installed, all the required tools have been installed and the project can be downloaded and flashed.

The project resides in a GitHub repo and can be forked to a local repository quite easily. Additionally, GitHub has a GUI installer for Windows to simplify the cloning process. See the [GitHub for Windows](https://windows.github.com/) site for the download. The repository is located at <https://github.com/Shamshel/Research>.

If the project will continue development with revision control through GitHub, the repository can be forked. GitHub has provided instructions for [forking a repository](https://help.github.com/articles/fork-a-repo/) that can be used to create a copy of the repository in a new account.

The project directory structure (as configured at the time of this writing) has the following structure.

1. Research
   1. EDK
      1. SDK
         1. PR\_Project
            1. Debug

src

* + - * 1. src
      1. PR\_Project\_bsp
         1. Microblaze\_0

code

include

lib

libsrc

bram\_v3\_02\_a

common\_v1\_00\_a

cpu\_v1\_15\_a

generic\_v1\_00\_a

hwicap\_v8\_01\_a

iic\_v2\_07\_a

spi\_v3\_05\_a

standalone\_v3\_10\_a

uartlite\_v2\_01\_a

xilisf\_v3\_02\_a

* + - 1. PR\_Project\_hw\_platform
      2. PR\_Project\_XADC
         1. Debug

src

* + - * 1. src
      1. PR\_Project\_XADC\_bsp
         1. microblaze\_0

code

include

lib

libsrc

bram\_v3\_02\_a

common\_v1\_00\_a

cpu\_v1\_15\_a

generic\_v1\_00\_a

hwicap\_v8\_01\_a

iic\_v2\_07\_a

standalone\_v3\_10\_a

sysmon\_v5\_03\_a

uartlite\_v2\_01\_a

* + - 1. PR\_Project\_XADC\_hw
    1. XPS
       1. PR\_Project
          1. \_xps
          2. data
          3. drivers
          4. etc
          5. hdl
          6. implementation
          7. pcores
          8. SDK
          9. synthesis
       2. PR\_Project\_XADC
          1. .Xil
          2. \_\_xps
          3. data
          4. etc
          5. hdl
          6. implementation
          7. pcores
          8. PR\_Project\_XADC.cache
          9. PR\_Project\_XADC.data
          10. PR\_Project\_XADC.runs
          11. PR\_Project\_XADC.srcs
          12. ring osc calibration runs
          13. SDK
          14. Synthesis

The two folders, PR\_Project and PR\_Project\_XADC, are two versions of the same project, PR\_Project was developed without the XADC controller, and is no longer developed but is useful as a reference. PR\_Project\_XADC was created from PR\_Project.

After the Xilinx ISE has been installed and the software repository has been cloned to the local machine, the project can be compiled and flashed to the FPGA. The target platform is the KC705 board based on the Kintex 7 FPGA from Xilinx. Follow these steps to connect the system to the host machine.

1. Touch a conductive or metal surface to discharge any accumulated static electricity. If necessary, connect to a static discharge wristband to reduce the chance of ESD.
2. The Kintex 7 has a custom power connector which is appropriately labeled **NOT PCIe POWER**. The only supplies that can be connected are the power brick supplied with the system, or a PCIe power port connected through the included adapter. Connect one of these supplies to the board, following the KC705 user manual, page 54 ([UG810](http://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf)), if necessary.
3. Connect the labeled UART and JTAG ports in the front of the board (with the PCIe cover plate) with the included USB mini and USB micro cables respectively.

Now that the board is connected, the system can be programmed, assuming no changes have yet been made to the board support package (which defines the FPGA interfaces, interconnects, and other variables), the system can be programmed from the SDK.

1. Start the Xilinx SDK (start -> All Programs -> Xilinx Design Tools -> ISE Design Suite xx.x -> EDK -> Xilinx Software Development Kit).
2. Once the SDK has started and is open, import the project. (File -> Import -> Existing Projects into Workspace, Next -> Select root directory (click Browse) -> navigate to the SDK folder of the repository -> Finish)
3. Select PR\_Project\_XADC\* folders, then Finish.
4. Build the project with the hammer symbol in the left half of the toolbar.
5. Flash the program to the board by selecting Xilinx Tools -> Program FPGA
6. Check that the correct bitstream is selected (by default, PR\_Project\_XADC\PR\_Project\_XADC.runs\counter\counter.bit)
7. Check that the correct BMM file is selected (by default, PR\_Project\_XADC\implementation\system\_bd.bmm)

Figure The build software icon.

1. Check that the correct ELF file is selected (by default, PR\_Project\_XADC\Debug\PR\_Project\_XADC.elf)

The program will begin flashing to the FPGA, this operation will take a minute or so. Once the flash has completed, start a serial terminal and connect to the local COM port. To determine which COM port, follow the following steps.

1. Click Start, type **Device Manager**.
2. Click Device Manager (perhaps under Control Panel).
3. In the Device Manager, expand the **Ports (COM & LPT)**.

Figure the flash FPGA icon.

1. Look for the **Silicon Labs CP210x USB to UART Bridge**.
2. Notice the COM port listed next to the entry.

The UART serial port is connected with 9600 Baud, 8-bit, no parity, 1 stop bit, no flow control. When the serial terminal is connected, the system can be reset by pressing the CPU RST (SW77), or by sending an ‘m’ character over the terminal. This should produce a menu with all the options available to the user. Refer to the C code or see the section in this document which pertains to the current functionality of each option.

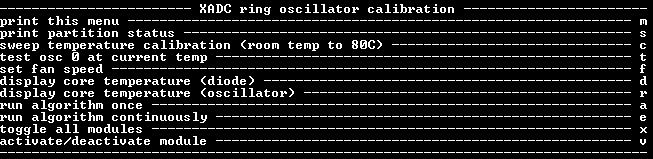


Figure 3 The PR\_Project main menu

# Directory Contents

# Option Functionalities

The following table will outline the functionalities of the various options in the PR\_Project main menu.

|  |  |  |
| --- | --- | --- |
| Menu Item | Corresponding Key | Function |
| print this menu | m (or any unassigned key) | Print main menu. |
| print partition status | s | Prints the information of all partitions. |
| sweep temperature calibration | c | Uses current partition activity set and begins sampling the oscillator modules on those partitions. The program then prints the frequencies of these partitions every 1 degree C as reported by the onboard temperature diode. |
| test osc 0 at current temp | t | Prints the current frequency of the oscillator closest to the onboard temperature diode. This number can be used to calibrate the frequency of the oscillator to the temperature of the diode. |
| set fan speed | f | Sets the fan intensity to fully on or fully off. The program will claim that the user can set a variable intensity, but it is only ever fully on or fully off. |
| display core temperature (diode) | d | Displays the result of sampling the onboard temperature diode. |
| display core temperature (oscillator) | r | Allows the user to select an oscillator to sample (in hex). When ‘q’ is pressed, the system then prints each sample of the selected oscillator’s output until the done bit is transmitted. |
| run algorithm once | a | Runs the floorplanning algorithm for one iteration. |
| run algorithm continuously | e | Runs the floorplanning algorithm indefinitely on a time interval set in the C code. |
| toggle all modules | x | Activates/deactivates all oscillator modules. |
| activate/deactivate module | v | Allows the user to select which module to activate/deactivate. |

# Description of Custom Cores

A number of custom cores have been developed for the use in this project. The first, and most complicated of which, is the Ring Oscillator core. This core went through a number of iterations and revisions until its functionality was as expected. The latest implementation, and the one used at the time of this writing, is located in EDK\XPS\PR\_Project\_XADC\pcores\reconfig\_modules\_source\rp\_counter. To open the ISE project, double click the file EDK\XPS\PR\_Project\_XADC\pcores\reconfig\_modules\source\rp\_counter\freq\_counter.xise. The frequency counter is based on a [ring oscillator](http://en.wikipedia.org/wiki/Ring_oscillator), a series of odd **not** gates connected input to output. The system then counts the number of oscillations. These oscillations are often implemented in VCO, but because the voltage is held constant, the variable is then temperature. The temperature changes alter the frequency in a decaying fashon. For more information, refer to the information and graphs available in \EDK\XPS\PR\_Project\_XADC\ring osc calibration runs\.

The ring oscillator core utilizes AXI interconnect system. The ring oscillator project is based on an auto-generated project from the XPS tool. To generate such a custom AXI peripheral, open the XPS tool and select Hardware -> Create and Import Peripheral, continue clicking next (create template for a new peripheral to an XPS project) select the required AXI interface type.

The ring oscillator project is based on such a template and utilizes two 32-bit wide-ports. The first port is utilized to control the activation and period of the ring oscillator. The second port is the output from the oscillator counter. The first AXI port consists of essentially two fields, a go bit, and oscillation count. The first bit of the 32-bit word (bit 31) is the go bit. The remaining bits (bits 30:0) are the oscillation count.

After transferring a configured word, and setting the go bit causes the ring oscillator to begin counting and continues until the number of FPGA master clock cycles denoted by oscillation count have passed. The second register is the output register from the counter. The output counter updates the number of oscillations on the output register every FPGA master clock cycle. When the number of FPGA master clock cycles denoted in the oscillation count field of the first register passes, the system sets the last bit of the second register (bit 31). This bit acts as the done bit, the remaining 31 bits (bits 30:0) are the number of ring oscillations detected in the period given. The following table shows the register bits described.

|  |  |  |
| --- | --- | --- |
| Register bit fields | | |
| reg0 | Go (bit 31) | Master Clock Oscillations (bit 30:0) |
| reg1 | Done (bit 31) | Result (bit 30:0) |

The ring oscillator module is implemented as a partially reconfigurable module configured in PlanAhead in the final project.

The other module developed for the project is the fan controller module. The fan controller module is much less interesting, and is not required for the operation of the system when the heat skink normally mounted on the FPGA die is removed (as it is at the time of this writing).

# Starting the Project from Scratch

## XPS

The XPS software is the suite that allows users to configure and generate the Microblaze soft processor core. Once generated, the processor core is then connected to the various peripherals that have been generated or created for the system to operate in software.

## PlanAhead

The PlanAhead software allows users to compile bitstreams for an FPGA and perform offline floorplanning. Here, the PR\_Project floorplans out the reconfigurable peripheral modules that are to be used later when implementing the partially reconfigurable peripherals. The partial bitstreams for these partially reconfigurable modules are stored in onboard flash memory for loading later.

## SDK

The SDK software is where the software for the Microblaze core is written and generated. Here, using the address scheme built in the XPS and PlanAhead software suites, the SDK can perform system IO and in the case of the PR\_Project, perform oscillation timing and counting. The SDK software suite generates ELF files, and is the last stage of the compilation and build process

## Toolchain Flow

The following outlines the steps taken to implement and run an application using the ISE toolchain.

1. Generate Microblaze processor using XPS
2. Add and connect peripherals to the Microblaze
3. Export a netlist for PlanAhead and a bsp for SDK
4. Import the netlist into PlanAhead
5. Floorplan the reconfigurable modules on the FPGA
6. Compile the resulting netlist into a bitstream
7. Create a software project for the system using SDK
8. Import an existing bsp, the one developed in XPS
9. Develop software
10. Generate an ELF file
11. Program the FPGA from the SDK using the ELF, bitstream, and bsp files

The following sections will outline these steps in detail.

## Generate Microblaze Processor

1. Start the XPS tool with Start -> All Programs -> Xilinx Design Tools -> EDK -> Xilinx Platform Studio
2. Click **Create New Project Using Base System Builder**
3. Create and select the desired source folder, select AXI System,
4. Select **Create a System for the Following Development Board (Pre-selected Device Info)**
5. Select Xilinx as the Board Vendor: **Xilinx** Board Name: **Kintex-7** **KC705** **Evaluation Platform** Board Revision: **C**
6. Select **Single Microblaze Processor System**
7. Select Optimization Strategy: Area
8. Set the Processor Frequency to 100 MHz
9. Set the Local Memory Size to 128 kB
10. Remove the following cores from the **Included Peripherals for microblaze\_0**:
    1. DDR3\_SDRAM
    2. DIP\_Switches\_8Bits
    3. Ethernet\_Lite
    4. LEDs\_8Bits
11. Set the following options in the RS232\_Uart\_1 module
    1. RS232\_Uart\_1: axi\_uartlite
    2. Baud Rate: 9600
    3. Data Bit Width: 8
    4. Parity: None
12. Click Finish
13. Once the project has generated with the default peripherals, double click the **QSPI\_FLASH**
14. Next to the **Use Startup** label, click the dropdown and select **DO NOT USE STARTUP Primitive**

## Add Peripherals and Connect Microblaze

### Add an XADC Peripheral

1. Expand the **Analog** tab in the **IP Catalog** section
2. Right click the **AXI XADC** and select **Make This IP Local**
   1. The AXI XADC needs to be modified to read the on-die temperature sensor. The modifications required are based on an example XADC project provided by Xilinx. These modifications are easiest to perform when the AXI folders are overridden with those located in the github repository. Open the project directory
   2. Open the pcores directory
   3. Notice the folders labeled **axi\_xadc\_v1\_00\_a**, **axi\_xadc\_v1\_00\_a\_axi\_lite\_ipif\_v1\_01\_a**, **axi\_xadc\_v1\_00\_a\_interrupt\_control\_v2\_01\_a**, **axi\_xadc\_v1\_00\_a\_proc\_common\_v3\_00\_a**
   4. These folders are the original AXI peripheral definitions, including HDL and MPD definitions. These definitions define two ports that form a differential input that the XADC peripheral uses to read the analog values read by the XADC daughterboard. The original array is defined as VAUXP and VAUXN, both of which are 16-bits wide. The modification is made in the definition of this array, two pins from each array are defined as two more differential arrays, VAUXN\_AMS and VAUXP\_AMS. The simplest way to make these changes is to copy the folders listed in the last step from the github repository. Copy the folders **axi\_xadc\_v1\_00\_a**, **axi\_xadc\_v1\_00\_a\_axi\_lite\_ipif\_v1\_01\_a**, **axi\_xadc\_v1\_00\_a\_interrupt\_control\_v2\_01\_a**, **axi\_xadc\_v1\_00\_a\_proc\_common\_v3\_00\_a** from the github repository’s pcore directory. These folders can be found in a pcore folder in the same location as navigated to above.
   5. Paste the folders selected above into the new project’s pcore folder and choose to merge all folders and override all files.
3. Once the XADC IP has been added to the pcores folder, expand the **Project Local PCores**, tab under the **IP Catalog**
4. Before adding the IP, click the **Rescan User IP Repositories** to update the component definition. Failing to do this will result in an incorrect **MHS** file connection
5. Under the **Analog** tab, right clock  **AXI XADC** and click **add IP**
6. When the configure IP box appears, check the box labeled **Enable Temperature Bus**, and **Include Interrupt**
7. Click **Ok** on this and the following window
8. One additional change needs to be implemented in the **MHS** file.

### Add reconfigurable peripherals

1. Copy the **reconfigurable\_peripherials\_v1\_00\_a** from the github repository to the **pcores** directory
2. Click the **Rescan User IP Repositories** button

Figure Rescan User IP Repositories

1. The reconfigurable peripherals IP should appear under **Project Local PCores** under the sub-category **USER**
2. Right click the reconfigurable peripherals module and select  **Add IP**
3. Click OK
4. Repeat steps 4 and 5 until 28 reconfigurable peripherals have been added.

### Add HWICAP interface

1. Expand the **FPGA Reconfiguration** section of **IP Catalog**
2. Right click the **FPGA Internal Configuration Access Port** IP and click **Add IP**
3. In the Core Config menu, click the **Instantiate STARTUP primitive in the HWICAP core**
4. Click OK on this and the next popup window

### Add Fan Speed Controller

1. Add the **fan\_controller\_v1\_00\_a** folder from the repository to the **pcores** folder
2. Click the **Rescan User IP Repositories** button to detect the newly installed fan controller
3. Right click the **FAN\_CONTROLLER** IP under the **USER** portion in the **IP Catalog**
4. Select **Add IP**

### Add Additional BRAM Modules

The C program developed later will likely require more BRAM than allocated in the core generator by default. The program in the repositories exceeded the code space of 128kB, so more BRAM was added for its continued development. Part of the reason that this code took so much space is the overhead involved with the Xilinx print library. If the program can be developed with a smaller code size, this may be unnecessary.

1. Expand the **Memory and Memory Controller** section of the **IP Catalog**
2. Right click the **LMB BRAM Controller** and select **Add IP**
3. Rename the module something more meaningful (since there will be a total of 4 BRAM controllers) such as **microblaze\_0\_d\_bram\_ctrl\_1**
4. Repeat step 3, but rename the module **microblaze\_0\_i\_bram\_ctrl\_1**
5. Right click the **Block RAM (BRAM) Block** in the **Memory and Memory Controller** section of the **IP Catalog**
6. Select **Add IP**
7. Connect the controllers to the BRAM block by clicking the “+” icons on the newly created **microblaze\_0\_d\_bram\_block**
8. In the drop down menu next to **PORTA**, select m**icroblaze\_i\_bram\_ctrl\_BRAM\_PORT**
9. Similarly, next to **PORTB**, select **microblaze\_d\_bram\_ctrl\_BRAM\_PORT**
10. Connect the controllers to the Microblaze processor by clicking the “+” next to the **microblaze\_0\_d\_bram\_ctrl\_1** and **microblaze\_0\_i\_bram\_crtl\_1**
11. Next to **SLMB** in both modules, assign memory controller **microblaze\_0\_d\_bram\_ctrl\_1** to **microblaze\_0\_dlmb** and **microblaze\_0\_i\_bram\_ctrl\_1** to **microblaze\_0\_ilmb** respectively

### Configure Base Addresses and Memory

The Microblaze AXI interface supports only 16 devices per bus. Thus, in order to support the necessary peripherals for the project, the AXI bus must be expanded.

1. The **AXI to AXI** connectors and **AXI interconnects** should have been added automatically, if for some reason they weren’t, or more interconnects are needed, refer to the following instructions
   1. Expand the **Bus and Bridge** section of the **IP Catalog**
   2. Right click the **AXI to AXI connector**  and select **Add IP**
   3. Add another **AXI to AXI connector** in this manner
   4. Right click the **AXI Interconnect** and select **Add IP**
   5. Add another **AXI Interconnect** in this manner
2. The most reliable way for the AXI interconnects to build correctly is to connect the **axi2axi\_connector**s to the primary Microblaze AXI bus. Do this by moving the cursor into the connection graphic just left of the **Bus interfaces** tab. Notice that a number of empty dots appear to indicate that connections have not been made in those locations. Connect each **axi2axi\_connector** to the first AXI bus by clicking the corresponding empty dots on the first AXI bus interface. The AXI bus interface is a green and white checkered column with the letters AXI at the top. The original AXI bus is the leftmost column
   1. It may be necessary to move another peripheral off the first AXI bus to make room for the connector as this is not done automatically, and each bus can only support 16 attachements
3. Double click the **axi2axi\_connector\_1**
4. In the popup window, change the **Slave AXI Number of ADDR Ranges** to 16
5. Double click the **axi2axi\_connector\_2**
6. In the popup window, change the **Slave AXI Number of ADDR Ranges** to 16
7. In the connection view next to the **Bus Interfaces** tab, from top to bottom, count the number of peripherals on the first AXI interface (represented as a vertical green line with **AXI** in green at the top)
8. When 16 connected peripherals have been counted, begin attaching peripherals to the second AXI interface until 16 devices have been attached and so on
9. Repeat this process, adding peripherals to each bus until all peripherals have been placed
10. If additional AXI interfaces are required, repeat the indented steps after step one
11. After the AXI connectors and interconnects have been added, switch to the **Address** tab.
12. Notice that there are two segments in the **Address** tab, **microblaze\_0’s Address Map** and **Unmapped Addresses**
13. Notice also a number of **axi2axi\_connector\_x** with address ranges which do not overlap with any peripherals and a number that are listed in the Unmapped Address space
14. The first step should be to instantiate the BRAM controller’s address space. In the **Size** column on the **microblaze\_0\_d\_bram\_ctrl\_1** and **microblaze\_0\_i\_bram\_ctrl\_1** select **128K**. This should automatically assign the addresses **0x0000 0000** to **0x0001 FFFF**. This address space assignment overlaps the original BRAM controllers. Remedy this by changing the address start space from **0x0000 0000** to **0x0002 0000**, this will automatically change the address end to **0x0003 FFFF**
15. Next, ensure that each **axi2axi\_connector\_x** is offset from each other connector by at least 64 kB (4 hexadecimal characters). The default for automatic addressing seems to actually be 128 kB. For example, if **axi2axi\_connector\_1** occupies 0x7250 0000 – 0x7250 FFFF, then the next **axi2axi\_connector\_1** should occupy 0x7252 0000 0x7252 FFFF. Ensure that when the address range is adjusted that the **size** section is set to 64K
16. After each **axi2axi\_connector** is offset, begin changing the base addresses to overlap the various **axi2axi\_connector** modules. For example, if **reconfigurable\_peripherials\_23** is unmapped, and there is an unoccupied connector **axi2axi\_connector\_2** at address **0x7266 0000**, change the base address of the unmapped **reconfigurable\_peripherials\_23** to **0x7266 0000**. If the peripheral is still listed as unmapped, refer back to the **Bus Interfaces** tab and connect the desired peripheral to the desired port. If, in the above example, **reconfigurable\_peripherials\_23** is listed as connected to **axi\_interconnect\_1** as shown in the **Bus Interfaces** tab, switch the green connector to create a junction at **axi\_interconnect\_2** to move the peripheral into mapped address space
17. Repeat these steps until there is no unmapped addresses remaining
18. To clean up the address space of unused AXI connectors, count the number of unused **axi2axi\_connector**s that do not overlap any other peripherals
19. Return to the **Bus Interfaces** tab and double click the **axi2axi\_connector** to which the unused connectors belong
20. Reduce the number next to **Slave AXI Number of ADDR Ranges** by the number counted in the previous steps, XPS will automatically remove the connectors from unmapped memory space

The following images show a completed address mapping, these may differ from implementation to implementation, so the following address map is intended to be used as an example.

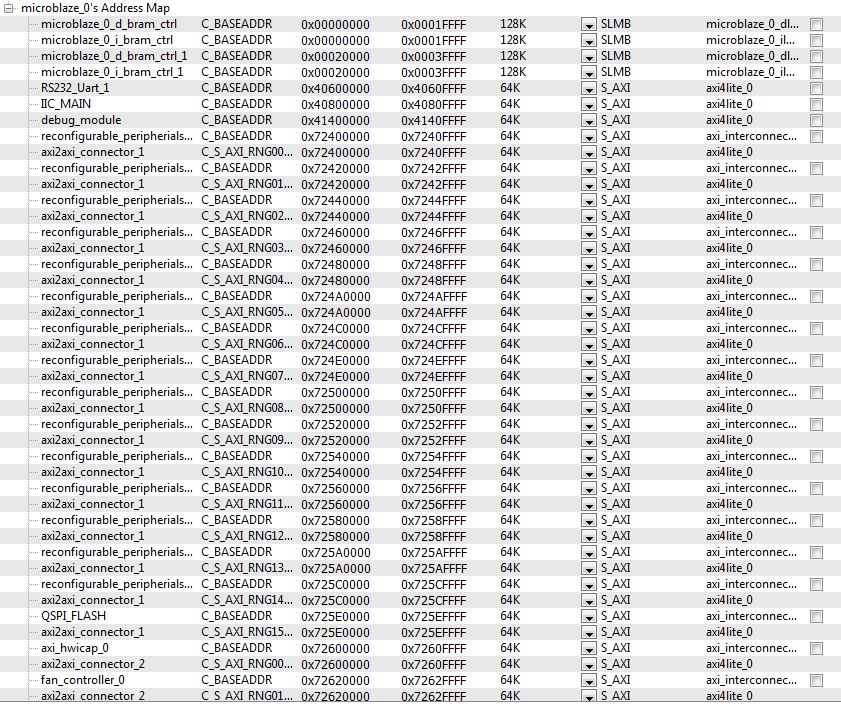


Figure 5 Microblaze address map (1/2)

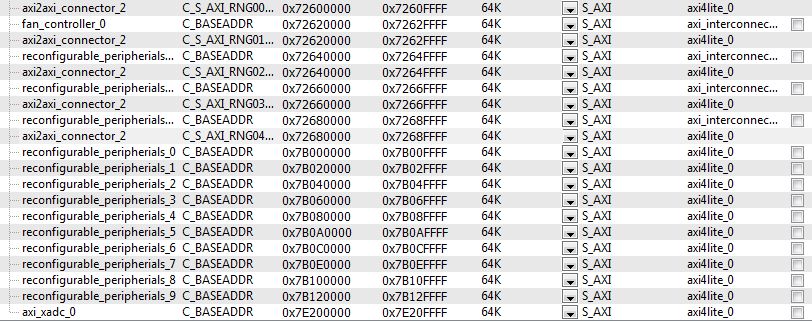


Figure 6 Microblaze address map (2/2)

### Set Clock Speed and Verify Clocks

1. Click **Hardware** on the toolbar
2. Select **Launch Clock Wizard**
3. Find **axi\_hwicap\_0 ICAP\_Clk** and change the clock source from **EXTERNAL** to **AUTO** and change the **Frequency** to **100 MHz**
4. At the bottom of the page, click **Validate Clocks**, this process will take about a minute

### Generate the netlist

Figure The Generate Netlist Icon

1. Click the **Hardware** tab at the top of the XPS window
2. Select **Generate Netlist** from the dropdown
3. Alternatively, click the **Generate Netlist** button in the toolbar

## Import netlist Into PlanAhead

1. Open PlanAhead by clicking Start, All Programs, Xilinx Design Tools, ISE Design Suite XX.X, PlanAhead
2. Once PlanAhead is opened, choose **Create New Project**
3. Continue by clicking Next
4. Name the project and select its location. The PlanAhead project can be placed in any directory, but the recommended location is in the same directory as the XPS project, refer to the directory structure above
5. Select the bullet next to **Post-synthesis Project**, as the project already has a netlist and needs only to be floor planned and synthesized
6. Select the check boxes next to **Enable Partial Reconfiguration**
7. Select Next
8. Select **Add Files…** and navigate to the XPS project’s directory
9. In the XPS project directory, change to the **Implementation** directory
10. Add all the files that end with **.ngc**, it may be necessary to repeat step eight and nine to add all the files as all the filenames won’t fit in the text box
11. click **Next**
12. In order to constrain the external ports required by the XADC peripheral, the UCF file included in the repository must be used (otherwise the UCF file can be constructed from scratch to suit a modified application). Copy **A portion** of the UCF file to the current project directory (if outside the repository directory)
    1. In windows explorer: locate the repository directory
    2. Navigate to the data directory of the project: EDK\XPS\PR\_Project\_XADC\data
    3. Copy all the text below the line ” TIMESPEC TS\_sys\_clk\_pin = PERIOD sys\_clk\_pin 200000 kHz;” (see appendix for constraints)
    4. Navigate to the new project data directory
    5. Paste the copied data below the line indicated above. These lines constrain the XADC pins and tell PlanAhead where the reconfigurable partitions are to be located on the FPGA die. At a minimum, the XADC pins must be constrained in the manner indicated, or the bitgen process will throw an unconstrained pin error. Similarly, do not simply copy the UCF file, it is quite likely that a XPS-generated constraint will be overridden and result in a similar unconstrained pin error.
13. In the **Add or Create Constraints** window, select **Add Files…**
14. Navigate to the XPS project directory and locate the **data** directory
15. Add the **system.ucf** file
16. It may be prudent to deselect the **Copy constraints files into project** check box to make changes in the system.ucf file immediately available to PlanAhead
17. Click Next
18. In the **Specify** window on the left, select **Boards**
19. In the **Filter** window, set the **Family** to **Kintex 7**
20. In theselection dialog at the bottom of the screen, select the **Kintex-7 KC705 Evaluation Platform** and click **Next**
21. Click **Finish**

## Open netlist

1. Click **Open Synthesized Design** under **Netlist Analysis** in the **Flow Navigator** on the left hand side of the screen
2. A popup window will appear denoting that the synthesized design is being opened, this can take several minutes
3. A final popup message will appear indicating a number of “critical messages” have arisen during opening the synthesized design (there should be 28, one for each partition), these are a result of the partial reconfiguration design flow and can be ignored, click **OK**

## Floorplan the FPGA

The design from XPS has been loaded and a rendering of the floorplan will appear on the right most side of the screen. It is now time to designate the size and location of each reconfigurable partition.

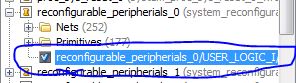
1. Begin by copying the ring oscillator hdl over to the newly created project
   1. In the github repository, locate and navigate into the **pcores** directory
   2. Copy the folder named **reconfig\_modules\_source**
   3. Navigate to the project directory and locate and navigate into the **pcores** directory
   4. Paste the copied folder into the **pcores** directory
2. Locate the **Netlist** view
3. Under **system** scroll down to the first occurrence of the reconfigurable peripherals; **reconfigurable\_peripherials**\_**0**, expand this instance

Figure The reconfigurable peripheral icon

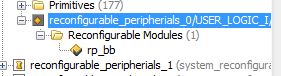
1. Right click the icon below the Nets and Primitives folders
2. Select **Set Partition…**
3. Select the bullet next to **is a reconfigurable Partition**
4. Name the module **rp\_bb** (reconfigurable partition – black box)
5. Select the bullet next to **Add this Reconfigurable Module as a black box without a netlist**
6. A summary page should appear, click **Finish**
7. Observe that the icon has changed to resemble the one below

Figure The reconfigurable peripheral icon after setting partition options

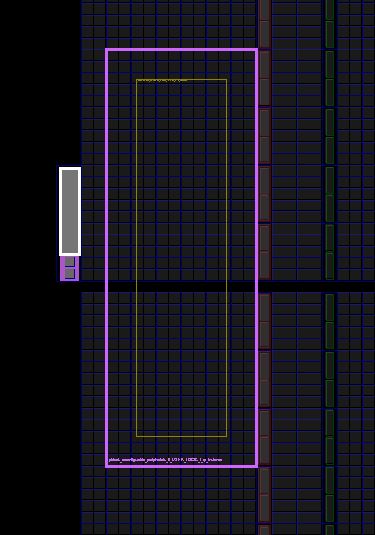
1. Right click the icon below the nets and primitives folders again and select **Add Reconfigurable Module**
2. Click **Next**
3. Name the new module **rp\_counter**
4. Click **Next**
5. Select the dots next to the box labeled **Top Netlist File:**
6. Navigate to the **pcores** directory
7. Navigate to **reconfig\_modules\_source/rp\_counter/**
8. Select the file **rp.ngc**
9. Click **Next**
10. No constraint file needs to be added, click **Next**
11. Click **Finish**
12. When the constraint file was copied earlier, it should have copied the constraint information for the reconfigurable partitions. However, if this failed, or if the pblock size needs to be changed, the following instructions will give some guidance on how they were floorplanned.
    1. Before setting the pblock size, it may be prudent to resize the **Device** window to provide more space to navigate. Grab the border between the **Netlist** window and the **Device** window and expand the **Device** window until the entire device is visible

Figure reconfigurable partition 0 pblock

* 1. The c program expects reconfigurable partition 0 to be located next to the internal temperature diode to ensure accurate calibration of the soft ring oscillators. Press **Ctrl** and move the mouse over the purple box in clock zone **X0Y3** (labeled in the bottom left corner of the clock region)and scroll up to zoom in
  2. Continue to zoom the view in the **Device** window until it resembles that on the right (without the purple rectangle on the right). The small purple block is an IO pad for communication with the XADC peripheral, the highlighted block immediately above it is the onboard temperature diode and is located approximately in the center of the die. The large, empty purple rectangle to the right of these two boxes is the partition pblock, which will be generated momentarily
  3. Right click the reconfigurable peripheral icon as shown above, select **Set Pblock Size**
  4. No window will pop up, but when the mouse is moved over the **Device** window, the mouse indicator changes to a cross-hair. In the bottom right of the PlanAhead window two boxes will indicate the location of the mouse and the corresponding FPGA resource under it. Click and drag starting at coordinate X38Y160 to X49Y194
  5. This will draw a pblock with the same dimensions as that shown above
  6. The location of this pblock is important, as the bitgen process will fail if the pblock does not begin and end on specific X coordinates. Pblocks can be arbitrarily tall, however, as long as they do not cross clock borders
  7. This process will need to be repeated for each subsequent reconfigurable partition in the Netlist view. This process will be quite tedious, and failure to begin and end on the appropriate X dimensions will result in a cryptic error message along the lines of “Illegal Pblock left/right bound”. A list of pblock dimensions are given in the table below

|  |  |  |
| --- | --- | --- |
| Partition name | Starting Coordinate | Ending Coordinate |
| reconfigurable\_peripherials\_0 | X38Y160 | X49Y194 |
| reconfigurable\_peripherials\_1 | X38Y310 | X49Y344 |
| reconfigurable\_peripherials\_2 | X38Y260 | X49Y294 |
| reconfigurable\_peripherials\_3 | X38Y210 | X49Y244 |
| reconfigurable\_peripherials\_4 | X38Y110 | X49Y144 |
| reconfigurable\_peripherials\_5 | X38Y60 | X49Y94 |
| reconfigurable\_peripherials\_6 | X38Y10 | X49Y44 |
| reconfigurable\_peripherials\_7 | X92Y310 | X103Y344 |
| reconfigurable\_peripherials\_8 | X92Y260 | X103Y294 |
| reconfigurable\_peripherials\_9 | X92Y210 | X103Y244 |
| reconfigurable\_peripherials\_10 | X92Y160 | X103Y194 |
| reconfigurable\_peripherials\_11 | X92Y110 | X103Y144 |
| reconfigurable\_peripherials\_12 | X92Y60 | X103Y94 |
| reconfigurable\_peripherials\_13 | X92Y10 | X103Y44 |
| reconfigurable\_peripherials\_14 | X0Y310 | X19Y344 |
| reconfigurable\_peripherials\_15 | X0Y260 | X19Y294 |
| reconfigurable\_peripherials\_16 | X0Y210 | X19Y244 |
| reconfigurable\_peripherials\_17 | X0Y160 | X19Y194 |
| reconfigurable\_peripherials\_18 | X0Y110 | X19Y144 |
| reconfigurable\_peripherials\_19 | X0Y60 | X19Y94 |
| reconfigurable\_peripherials\_20 | X0Y10 | X19Y44 |
| reconfigurable\_peripherials\_21 | X108Y310 | X121Y344 |
| reconfigurable\_peripherials\_22 | X108Y260 | X121Y294 |
| reconfigurable\_peripherials\_23 | X108Y210 | X121Y244 |
| reconfigurable\_peripherials\_24 | X108Y160 | X121Y194 |
| reconfigurable\_peripherials\_25 | X108Y110 | X121Y144 |
| reconfigurable\_peripherials\_26 | X108Y60 | X121Y94 |
| reconfigurable\_peripherials\_27 | X108Y10 | X121Y44 |

## Compile netlist into bitstream

Before the compilation can begin, the system.bmm needs to be linked by modifying the default run configuration.

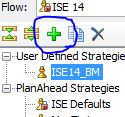
1. Click Tools -> Options
2. In the left window, select the **Strategies** icon
3. Click the drop down next to **Flow** and select **ISE 14** under **Implementation Strategies**
4. Select **ISE Defaults** under **PlanAhead Strategies** in the window immediately below the **Flow** dropdown
5. Click the **+** symbol in the toolbar between the windows as shown in the image to the right

Figure The add strategy icon

1. Name the strategy **ISE14\_BM** as shown in the image
2. Click OK
3. In the **Options** window to the right scroll down to **More Options\*** under **Translate (ngdbuild)**

Figure The Run Implementation icon

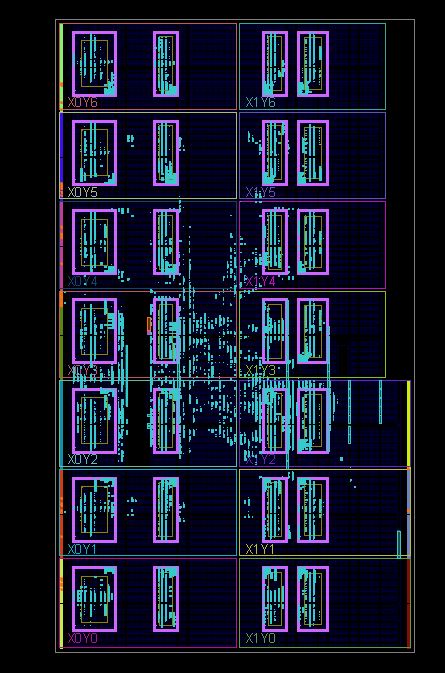
1. Type the following into the **More Options\*** box: **-bm ../../../implementation/system.bmm**
2. Click **Apply** and then **OK**
3. At the bottom of the PlanAhead window, in the **Design Runs** tab, right click the first item and select **Implementation Run Properties** at the top of the right-click menu. This will cause the **Implementation Run Properties** window to appear above the **Design Runs** box, if it was not already present.
4. First, rename the implementation run to **blanking**. This run will consist of empty (black box) placeholder partitions in each reconfigurable partition
5. In the **Implementation Run Properties** window notice the tabs at the bottom beginning with **General | Attributes | Options | …** click **Options**
6. Next to **Strategy** click the dropdown and select the newly created strategy **ISE14\_BM**
7. Next, click the right black arrow next to the rightmost tab on the bottom of the **Implementation Run Properties** window. To scroll to the right. Select **Partitions**
8. In the **Partitions** tab of the  **Implementation Run Properties** window, ensure that each partition is set to **rp\_bb** or the name of the black box in each partition
9. Click **Apply**
10. Right click in the **Design Runs** box where the newly configured blanking run is visible, and select **Create Runs…**
11. A window will pop up to guide the run creation process. Click **Next**
12. Rename the new run **rp\_counter**, ensure that the part is **xc7k325tffg900-2** and that the strategy selected is **ISE14\_BM**, and click **Next**
13. In the **Launch Options** window select the radio button next to **Do not launch now**
14. Click **Next** and then **Finish**
15. The blanking configuration is for future use. The rp\_counter configuration and the ring oscillators are intended to simulate blanking through deactivation. Right click the **rp\_counter** implementation run and select **Make Active**

Figure The FPGA floorplan and usage after the Implementation completes

1. Click the **Run Implementation** icon in the toolbar
2. The implementation process takes tens of minutes to complete. The run will halt after the completion of the Place and Route (PAR) stage and await further user input
3. After the PAR process has completed, the system can be compiled into a raw bitstream. The bitstream is the file used to program the FPGA with the processor and peripherals that will enable experimentation. Begin by noticing the **Flow Navigator** on the left most side of the screen
4. At the bottom of the **Flow Navigator** is a sub-menu labeled **Program and Debug**. Expand the **Program and Debug** section
5. Click the **Generate Bitstream** item in the sub-menu. PlanAhead will then begin generating the bitstream file for use in programming the project. The bitstream generation process can take upwards of two hours per run

## Export netlist and bsp

Once the bitstream has been generated, it is possible program the FPGA with the processor and peripherals. These components would not do anything, however, as no ELF file has been generated to instruct the processor how to operate. In order to combine the ELF file and the netlist file, it is necessary to export the build results to the Xilinx Software Development Kit (SDK).

1. Open the XPS software as in the steps before PlanAhead and select **Open Recent Project** from the main menu
2. From the resulting drop down, select the name of the project as it was created earlier

Figure the Export to SDK icon

1. Once the project has processed and opened, click the **Export to SDK** button
2. A new window will appear, deselect the option labeled **Include bitstream and BMM** file In order for XPS to generate these files, it would have to be capable of performing the floorplanning steps that were done manually in previous steps

## Import the BSP into XPS

Now that the XPS project has been exported, it has exposed the XML files required for the SDK to generate a **hardware platform specification** and a **board support package** (BSP).

1. In the SDK, select **File -> new -> Board Support Package**
2. A popup will appear indicating that no hardware platforms exist in the current workspace, select **Specify**
3. Name the project, the project name in the GitHub account is **PR\_Project\_hw\_platform** to indicate that the project is the hardware platform specification. It is therefore suggested that the naming convention follow **$Project\_Name$\_hw\_platform**
4. Under **Target Hardware Specification** select **Browse**
5. Navigate back to the project root directory
6. Select the newly generated folder called **SDK** -> **SDK\_Export -> hw** and select **system.xml**
7. Click **Finish**
8. The **Board Support Package Project** wizard then appears. Following the naming convention, change the project name to **$Project\_Name$\_bsp**
9. Ensure that **standalone** is selected in the **Board Support Package OS** window
10. Click **Finish**
11. The **Board Support Package Settings** wizard will appear
12. Under **Supported Libraries** select the box next to **xilisf**

## Create Software Project in SDK

Now that the board support packages have been created, the software project can be created.

1. Select **File -> new** **-> Application Project**
2. Name the project following the convention given, simply **$Project\_Name$**
3. Ensure that the **OS Platform** under **Target Software** is set to **standalone** under the dropdown
4. An error message may appear at the top of the window indicating that a bsp already exists. Next to **Board Support Package** select the radio button next to **Use existing** and select the BSP created in the previous section
5. Select **Empty Application** as the project template under the **Available Templates** window

## Edit Linker Script

Since the design uses an expanded block RAM (BRAM) space, it is necessary to correct the linker script.

1. Expand the newly created application project
2. Expand the **src** folder
3. Double click the **lscript.ld** to open the Linker Script configuration window
4. Change the stack and heap sizes. Change the hex numbers next to **Stack Size** and Heap **Size** to **0x800**
5. In the **Section to Memory Region Mapping** window, all but the following from the default option, **microblaze\_0\_i\_bram\_ctrl\_microblaze\_0\_d\_bram\_ctrl** to the other memory region, m**icroblaze\_0\_i\_bram\_ctrl\_1\_microblaze\_0\_d\_bram\_ctrl\_1**
   1. .text
   2. .init
   3. .fini

## Develop C Software

The C sources to drive the Microblaze processor can now be incorporated into the project. The next few steps will walk through importing the files from the GitHub into a new software project.

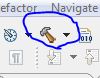
1. In the GiHub repository, navigate to **EDK -> SDK -> PR\_Project\_XADC -> src**
2. Ensure that the following files are present in the directory
   1. main.c
   2. platform.c
   3. platform.h
   4. platform\_config.h
3. In the SDK software, expand the software project folder
4. Right click the **src** folder and select **Import…**
5. A popup will appear, under the **General** folder select **File System**
6. Click **Next**
7. In the next window, select **Browse** which is in line with the empty box of text next to the label **From directory:**
8. Browse to the github source directory (**EDK -> SDK -> PR\_Project\_XADC -> src**) and click **OK**
9. The source directory from the github will then be listed in the window below the **From directory:** line. In the right window, select the following files.
   1. main.c
   2. platform.c
   3. platform.h
   4. platform\_config.h
10. Click **Finish**, although the project should have been created with an empty **src** file, if any files need to be overridden from the default auto-generated files, click **Yes**. Overriding these files with the **Import** dialog will replace the files completely, causing a potential for loss of data.
11. Now that the files have been imported, they can be built into an **ELF** file. The **ELF** file is the binary file that the Microblaze will execute in runtime. With the software project selected in the **Project Explorer**, click the **build** icon.
12. If the build fails because of missing header files, check that the files were generated by expanding the **$Project\_Name$\_bsp** project and then expanding the **microblaze\_0** folder and then expanding the **include** folder. If the **include** folder is not populated with headers, then the **$Project\_Name$\_bsp** needs to be reassociated

Figure The Build icon

* 1. Right click the bsp project and select **Delete**
  2. Check the box next to **Delete project contents on disk**
  3. Click **OK**
  4. A popup will appear warning about projects failing to build due to broken dependencies, click **OK**
  5. Right click in the **Project Explorer** and select **new -> Project…**
  6. Select the **Board Support Package**
  7. Click **Next**
  8. Rename the project identically to how it was named before it was deleted in the **Project Name** field
  9. Select the correct **Hardware Platform** from the dropdown
  10. Select **standalone** in the box under **Board Support Package OS**
  11. Click **Finish**
  12. Repeat step 11, and build the project

## Program the FPGA

Now that the ELF and bit files have been generated, the FPGA can be programmed.

1. Connect the FPGA to the custom 6-pin power supply, refer to the KC705 user manual, page 54 ([UG810](http://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf))

Figure Program FGPA icon

1. Connect the JTAG and UART ports to the computer over USB
2. Power on the board by switching on the power switch in the rear of the board next to the power supply connector
3. Confirm the UART is connected to the machine by connecting to it with a serial terminal. The terminal used thus far for this project is [Tera Term](http://ttssh2.osdn.jp/index.html.en). The device will appear as **Silicon Labs CP210x USB to UART Bridge**
4. Click the **Program FPGA** icon
5. A popup window should appear that allows you to select the bitstream, BMM, and ELF files. Select **Browse** next to the **Bitstream:** text box
6. Navigate to the PlanAhead folder and open the **$Project\_Name$.runs** folder
7. Open the **rp\_counter** folder
8. If the bitstream was generated successfully, a file named **rp\_counter.bit** should exist in this folder, select it and click **open**. A number of other partial bitstreams will have been generated for each partition, do not select these as they only define the portions of the design in the reconfigurable areas. The **rp\_counter.bit** file will contain all the information the FPGA needs to instantiate each reconfigurable partition
9. Select the **Browse** button next to **BMM File:**
10. Navigate to the XPS project directory
11. In the **implementation** folder, there is a file named **system\_bd.bmm**, select it and press **Open**
12. In the **Sofware Configuration** section, click the cell next to **microblaze\_0** in the **Processor** column. The cell should expand into a drop-down. If the software project completed successfully and generated an **ELF** file, select the **ELF** file, the other options are **bootloop** and **Browse…**, if the **ELF** file is not listed, but it was generated, the **Browse…** option can be used to point the programmer to the **ELF** file

# Future Work

# Appendix

## UCF Constraints

NET "QSPI\_FLASH\_IO0" LOC = P24;

NET "QSPI\_FLASH\_IO1" LOC = R25;

NET "QSPI\_FLASH\_SS" LOC = U19;

NET "axi\_xadc\_0\_VAUXP\_AMS\_pin[0]" LOC = J23;

NET "axi\_xadc\_0\_VAUXP\_AMS\_pin[0]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_VAUXN\_AMS\_pin[0]" LOC = J24;

NET "axi\_xadc\_0\_VAUXN\_AMS\_pin[0]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_VAUXP\_AMS\_pin[1]" LOC = L22;

NET "axi\_xadc\_0\_VAUXP\_AMS\_pin[1]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_VAUXN\_AMS\_pin[1]" LOC = L23;

NET "axi\_xadc\_0\_VAUXN\_AMS\_pin[1]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_CONVST\_pin" LOC = AC27;

NET "axi\_xadc\_0\_CONVST\_pin" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_MUXADDR\_pin[0]" LOC = A11;

NET "axi\_xadc\_0\_MUXADDR\_pin[0]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_MUXADDR\_pin[1]" LOC = A12;

NET "axi\_xadc\_0\_MUXADDR\_pin[1]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_MUXADDR\_pin[2]" LOC = A13;

NET "axi\_xadc\_0\_MUXADDR\_pin[2]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_MUXADDR\_pin[3]" LOC = A15;

NET "axi\_xadc\_0\_MUXADDR\_pin[3]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_MUXADDR\_pin[4]" LOC = A16;

NET "axi\_xadc\_0\_MUXADDR\_pin[4]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[0]" LOC = A17;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[0]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[1]" LOC = A18;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[1]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[2]" LOC = A20;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[2]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[3]" LOC = A21;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[3]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[4]" LOC = A22;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[4]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[5]" LOC = A25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[5]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[6]" LOC = A26;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[6]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[7]" LOC = A27;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[7]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[8]" LOC = A28;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[8]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[9]" LOC = A30;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[9]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[10]" LOC = AA20;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[10]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[11]" LOC = AA30;

NET "axi\_xadc\_0\_TEMP\_OUT\_pin[11]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[0]" LOC = AB20;

NET "axi\_xadc\_0\_ALARM\_pin[0]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[1]" LOC = AB27;

NET "axi\_xadc\_0\_ALARM\_pin[1]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[2]" LOC = AB29;

NET "axi\_xadc\_0\_ALARM\_pin[2]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[3]" LOC = AB30;

NET "axi\_xadc\_0\_ALARM\_pin[3]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[4]" LOC = AC22;

NET "axi\_xadc\_0\_ALARM\_pin[4]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[5]" LOC = AC24;

NET "axi\_xadc\_0\_ALARM\_pin[5]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[6]" LOC = AC25;

NET "axi\_xadc\_0\_ALARM\_pin[6]" IOSTANDARD = LVCMOS25;

NET "axi\_xadc\_0\_ALARM\_pin[7]" LOC = AC26;

NET "axi\_xadc\_0\_ALARM\_pin[7]" IOSTANDARD = LVCMOS25;

INST "reconfigurable\_peripherials\_0/reconfigurable\_peripherials\_0/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_0\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_0\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y160:SLICE\_X49Y194;

INST "reconfigurable\_peripherials\_1/reconfigurable\_peripherials\_1/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_1\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_1\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y310:SLICE\_X49Y344;

INST "reconfigurable\_peripherials\_3/reconfigurable\_peripherials\_3/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_3\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_3\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y210:SLICE\_X49Y244;

INST "reconfigurable\_peripherials\_4/reconfigurable\_peripherials\_4/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_4\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_4\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y110:SLICE\_X49Y144;

INST "reconfigurable\_peripherials\_2/reconfigurable\_peripherials\_2/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_2\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_2\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y260:SLICE\_X49Y294;

INST "reconfigurable\_peripherials\_5/reconfigurable\_peripherials\_5/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_5\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_5\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y60:SLICE\_X49Y94;

INST "reconfigurable\_peripherials\_6/reconfigurable\_peripherials\_6/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_6\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_6\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X38Y10:SLICE\_X49Y44;

INST "reconfigurable\_peripherials\_7/reconfigurable\_peripherials\_7/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_7\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_7\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y310:SLICE\_X103Y344;

INST "reconfigurable\_peripherials\_8/reconfigurable\_peripherials\_8/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_8\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_8\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y260:SLICE\_X103Y294;

INST "reconfigurable\_peripherials\_10/reconfigurable\_peripherials\_10/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_10\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_10\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y160:SLICE\_X103Y194;

INST "reconfigurable\_peripherials\_11/reconfigurable\_peripherials\_11/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_11\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_11\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y110:SLICE\_X103Y144;

INST "reconfigurable\_peripherials\_12/reconfigurable\_peripherials\_12/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_12\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_12\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y60:SLICE\_X103Y94;

INST "reconfigurable\_peripherials\_13/reconfigurable\_peripherials\_13/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_13\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_13\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X92Y10:SLICE\_X103Y44;

INST "reconfigurable\_peripherials\_9/reconfigurable\_peripherials\_9/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_9\_USER\_LOGIC\_I\_rp\_instance\_1";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_9\_USER\_LOGIC\_I\_rp\_instance\_1" RANGE=SLICE\_X92Y210:SLICE\_X103Y244;

INST "reconfigurable\_peripherials\_14/reconfigurable\_peripherials\_14/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_14\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_14\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y310:SLICE\_X19Y344;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_14\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y124:DSP48\_X0Y137;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_14\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y124:RAMB18\_X0Y137;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_14\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y62:RAMB36\_X0Y68;

INST "reconfigurable\_peripherials\_15/reconfigurable\_peripherials\_15/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_15\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_15\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y260:SLICE\_X19Y294;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_15\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y104:DSP48\_X0Y117;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_15\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y104:RAMB18\_X0Y117;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_15\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y52:RAMB36\_X0Y58;

INST "reconfigurable\_peripherials\_16/reconfigurable\_peripherials\_16/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_16\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_16\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y210:SLICE\_X19Y244;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_16\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y84:DSP48\_X0Y97;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_16\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y84:RAMB18\_X0Y97;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_16\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y42:RAMB36\_X0Y48;

INST "reconfigurable\_peripherials\_17/reconfigurable\_peripherials\_17/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_17\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_17\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y160:SLICE\_X19Y194;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_17\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y64:DSP48\_X0Y77;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_17\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y64:RAMB18\_X0Y77;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_17\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y32:RAMB36\_X0Y38;

INST "reconfigurable\_peripherials\_18/reconfigurable\_peripherials\_18/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_18\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_18\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y110:SLICE\_X19Y144;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_18\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y44:DSP48\_X0Y57;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_18\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y44:RAMB18\_X0Y57;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_18\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y22:RAMB36\_X0Y28;

INST "reconfigurable\_peripherials\_19/reconfigurable\_peripherials\_19/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_19\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_19\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y60:SLICE\_X19Y94;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_19\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y24:DSP48\_X0Y37;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_19\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y24:RAMB18\_X0Y37;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_19\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y12:RAMB36\_X0Y18;

INST "reconfigurable\_peripherials\_20/reconfigurable\_peripherials\_20/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_20\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_20\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X0Y10:SLICE\_X19Y44;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_20\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X0Y4:DSP48\_X0Y17;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_20\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB18\_X0Y4:RAMB18\_X0Y17;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_20\_USER\_LOGIC\_I\_rp\_instance" RANGE=RAMB36\_X0Y2:RAMB36\_X0Y8;

INST "reconfigurable\_peripherials\_21/reconfigurable\_peripherials\_21/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_21\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_21\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y310:SLICE\_X121Y344;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_21\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y124:DSP48\_X4Y137;

INST "reconfigurable\_peripherials\_22/reconfigurable\_peripherials\_22/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_22\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_22\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y260:SLICE\_X121Y294;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_22\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y104:DSP48\_X4Y117;

INST "reconfigurable\_peripherials\_23/reconfigurable\_peripherials\_23/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_23\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_23\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y210:SLICE\_X121Y244;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_23\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y84:DSP48\_X4Y97;

INST "reconfigurable\_peripherials\_24/reconfigurable\_peripherials\_24/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_24\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_24\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y160:SLICE\_X121Y194;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_24\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y64:DSP48\_X4Y77;

INST "reconfigurable\_peripherials\_25/reconfigurable\_peripherials\_25/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_25\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_25\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y110:SLICE\_X121Y144;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_25\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y44:DSP48\_X4Y57;

INST "reconfigurable\_peripherials\_26/reconfigurable\_peripherials\_26/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_26\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_26\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y60:SLICE\_X121Y94;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_26\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y24:DSP48\_X4Y37;

INST "reconfigurable\_peripherials\_27/reconfigurable\_peripherials\_27/USER\_LOGIC\_I/rp\_instance" AREA\_GROUP = "pblock\_reconfigurable\_peripherials\_27\_USER\_LOGIC\_I\_rp\_instance";

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_27\_USER\_LOGIC\_I\_rp\_instance" RANGE=SLICE\_X108Y10:SLICE\_X121Y44;

AREA\_GROUP "pblock\_reconfigurable\_peripherials\_27\_USER\_LOGIC\_I\_rp\_instance" RANGE=DSP48\_X4Y4:DSP48\_X4Y17;