

## Reference Manual

RedCape Board  
Version A01

FPGA Cape board for BeagleBone with peripheral  
connectors and sensors

July 2014

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The RedCape	1
1.2	Architecture	3
1.3	Board Layout	4
1.4	Getting Started	4
1.5	Stand Alone	6
<b>2</b>	<b>Sensors</b>	<b>8</b>
2.1	Sensors	8
2.2	Other I2C devices	9
2.2.1	ID EEPROM	9
2.2.2	Power Monitor	10
2.2.3	Real Time Clock	10
2.2.4	Quad Clock Generator	11
2.3	I2C bus connection	11
<b>3</b>	<b>Cyclone V</b>	<b>13</b>
3.1	Configuration Options	13
3.1.1	USB Blaster	15
3.1.2	Jrunner	16
3.1.3	Active Serial with USB Blaster	18
3.1.4	Active Serial with Jamplayer	19
3.2	Debbuging	20
3.2.1	LEDs	20
3.2.2	Push Button	21
3.2.3	Clock	22
3.3	Peripherals	22
3.3.1	JTAG	22
3.3.2	PFC – 15 (RaspBerry Pi Cam)	23
3.3.3	Pmod	24
3.3.4	SATA	26
3.3.5	SDI Video	27

3.3.6 P8 expansion port	28
3.4 Sensors and other devices	29
3.4.1 Sensors	29
3.4.2 Quad Clock Generator	29
3.5 Connection to the BeagleBone	31
3.6 Memory	32
3.6.1 DDR3 SDRAM Memory	32
3.6.2 NAND Flash memory	32
4 Code Examples	34
4.1 Use of I <sup>2</sup> C devices in the RedCape, accelerometer	34
4.2 PWM from Cyclone V	36
5 Appendix	40
5.1 Connection Tables	40
5.2 Schematics	46

## List of figures

<i>Figure 1.1</i>	<i>Block diagram of the RedCape</i>
<i>Figure 1.2</i>	<i>The RedCape</i>
<i>Figure 1.3</i>	<i>RedCape Mounted on the Beaglebone</i>
<i>Figure 2.1</i>	<i>Resistors R55 and R56 for I<sup>2</sup>C with the Cyclone V</i>
<i>Figure 2.2</i>	<i>Block diagram of the I<sup>2</sup>C devices</i>
<i>Figure 2.3</i>	<i>Result of i2cdetect command</i>
<i>Figure 3.1</i>	<i>Dip Switch in the RedCape</i>
<i>Figure 3.2</i>	<i>USB Blaster connected to the JTAG Port in the RedCape</i>
<i>Figure 3.3</i>	<i>Flash Memory Programming for Active Serial Configuration</i>
<i>Figure 3.4</i>	<i>LEDS and Buttons of the RedCape</i>
<i>Figure 3.5</i>	<i>JTAG Connector on the RedCape</i>
<i>Figure 3.6</i>	<i>PFC -15 connector for Raspberry Pi Cam</i>
<i>Figure 3.7</i>	<i>Pmod -12 Connector</i>
<i>Figure 3.8</i>	<i>SATA connectors in the RedCape</i>
<i>Figure 3.9</i>	<i>SDI connectors in the RedCape</i>
<i>Figure 3.10</i>	<i>P8 expansion port</i>
<i>Figure 3.11</i>	<i>Window of the Clock Builder Desktop</i>
<i>Figure 3.12</i>	<i>GPMC Block Diagram</i>
<i>Figure 3.13</i>	<i>Memories in the RedCape</i>
<i>Figure 4.1</i>	<i>Results from i2cdetect command</i>
<i>Figure 4.2</i>	<i>Output of demo for LIS331</i>
<i>Figure 4.3</i>	<i>Eclipse window with the code for the LIS331 demo</i>
<i>Figure 4.4</i>	<i>Block Diagram of PWM control example</i>
<i>Figure 4.5</i>	<i>Pin Planner in Quartus</i>
<i>Figure 4.6</i>	<i>LabVIEW GUI to control the PWM signal</i>
<i>Figure 4.7</i>	<i>Visualisation of PWM in user LEDs</i>

## Acronyms

EMC	Electromagnetic compatibility
FPGA	Field Programmable Gate Array
GPIO	General purpose input/output
GPMC	General purpose memory controller
GPS	Global positioning system
IMU	Inertial measurement unit
I <sup>2</sup> C	Inter integrated circuit
JTAG	Joint Test Action Group
LED	Light emitting diode
PWM	Pulse with modulation
SATA	Serial AT attachment
SDI	Serial Digital Interface
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus
VHDL	VHSIC hardware description language

# 1 - Introduction

## *1.1 - The RedCape*

The RedCape is a BeagleBone cape (<http://www.beagleboard.org/bone>) that features the latest generation of Field Programmable Gate Array (FPGA) from Altera together with a host of sensors and memory devices. The FPGA can behave as a processor, as a DSP, as a memory or as many other devices, allowing the user to generate a multitude of electronic designs and implement them. The Altera Cyclone V GX FPGA can work at high frequencies and create large-scale parallel designs, making it a powerful tool for a wide number of applications like large volume data processing. This GX version of the new generation of Cyclone devices offers high-performance 6 GHz transceiver channels and a dedicated hard memory controller to control DDR3 memories.

The RedCape has two large memory devices for storing information, a volatile DDR3 4Gb and a non volatile 8Gb NAND device. These devices make storing data possible for later processing or just to keep a log of the sensors measurements. The amount of memory available makes image processing applications possible with high quality pictures and/or videos. A separate flash memory is available that keeps the configuration of the FPGA allowing the system to program itself upon startup.

The use of the FPGA configuration flash memory is but one method to program the FPGA. It is also possible to configure the FPGA using the BeagleBone processor and an external ByteBlaster USB cable. The FPGA communicates with the processor of the BeagleBone through its general-purpose memory interface providing a high performance interface for exchanging data and instructions. Through 44 I/O lines a user can interface the FPGA to a large variety of equipment such as stepper motors, encoders, image sensors.

In addition to the FPGA the RedCape has been equipped with a variety of sensors that will be very helpful for many applications. Further sensors can be added to the RedCape by using standard PMOD (add reference) modules, RedCape has two such connectors suitable for a wide range of additional sensors and I/O devices such as GPS and wireless communications.

The RedCape offers a great way to get started with FPGA's and there are many example projects available. Although the RedCape was designed as an interface to the BeagleBone it is fully functional stand-alone device.

## 1.2 - Architecture

Figure 1.2 shows a block diagram with the architecture of the RedCape and its inter- and intraconnections. For completeness, the BeagleBone is also shown connected to the RedCape using its general-purpose parallel bus.

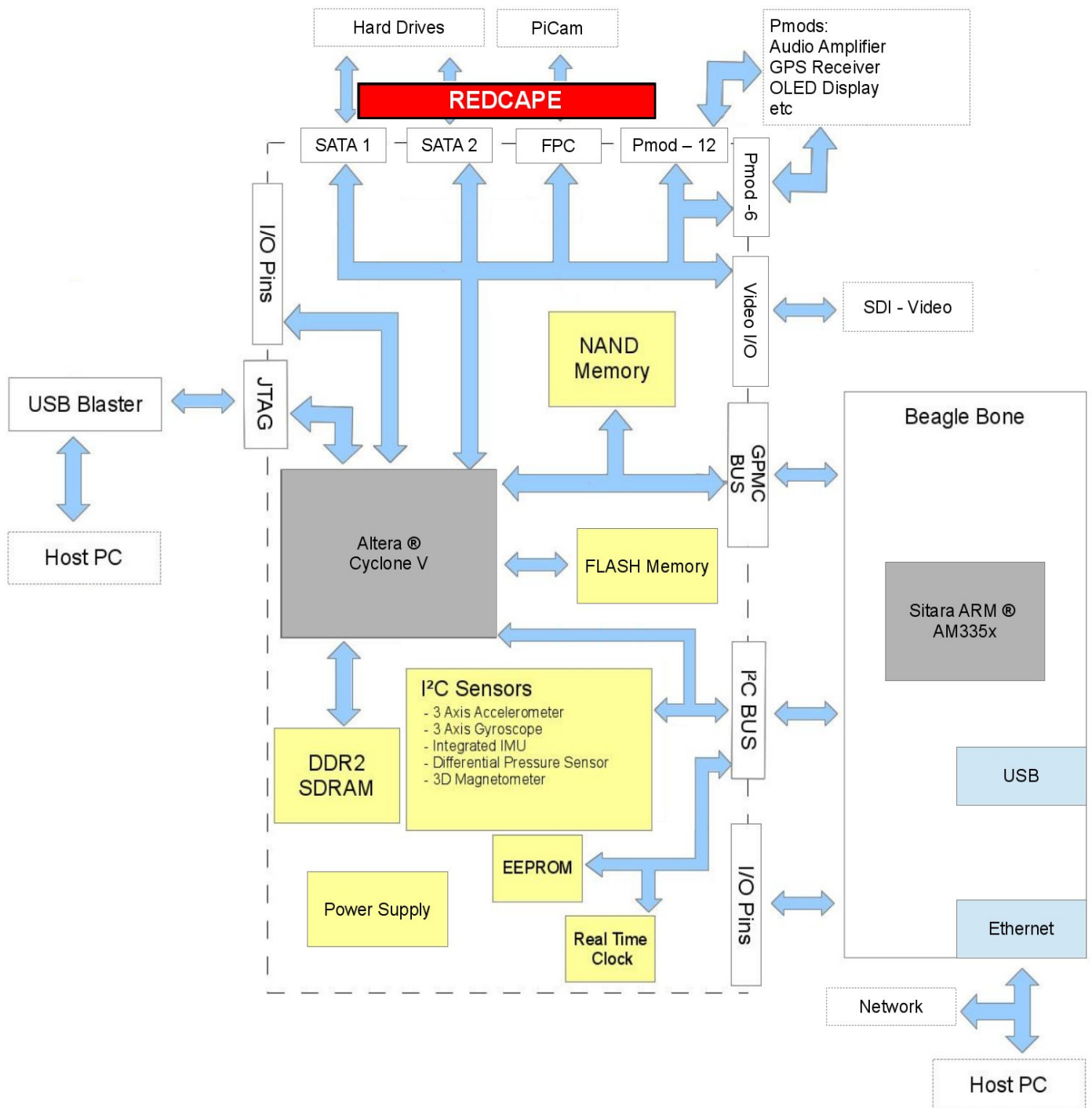


Figure 1.1 – Block diagram of the RedCape



## 1.3 - Board Layout

The next images show the components of the RedCape:

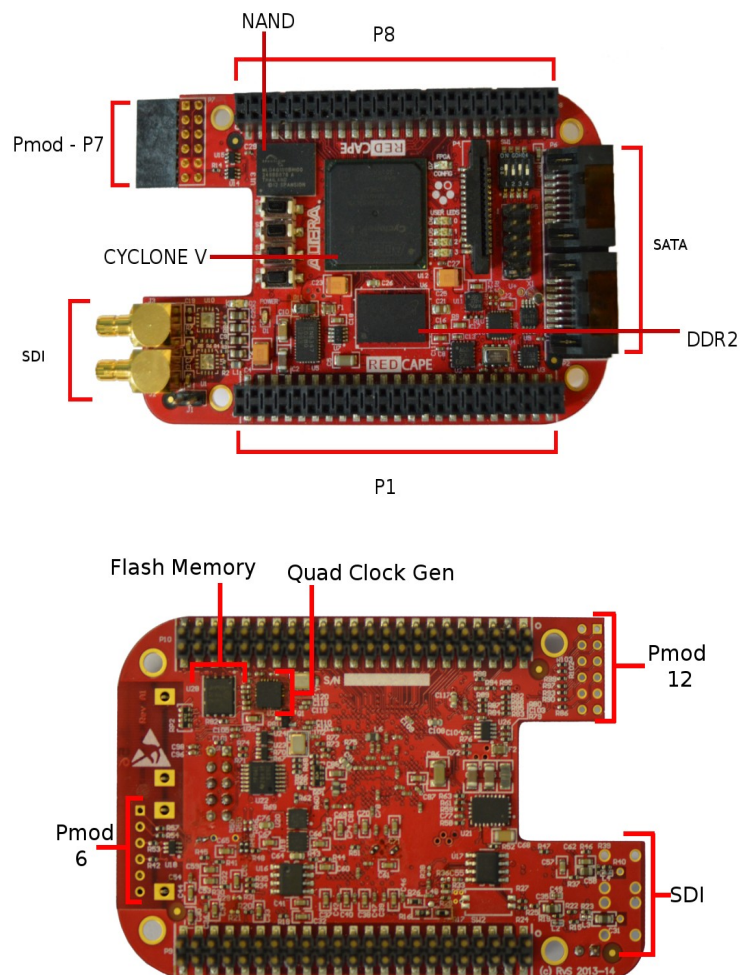


Figure 1.2 – The RedCape

## 1.4 - Getting Started

Before using the RedCape it is important to set up the BeagleBone correctly. This is particularly important when using the BeagleBone Black.

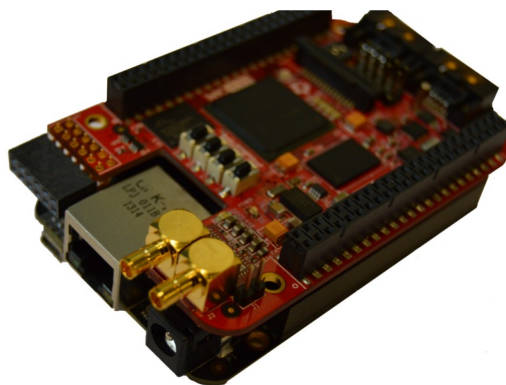
In this manual we shall assume that the user uses the default Linux distribution, Angstrom. Other Linux distributions can be used and should require minimum

changes in the steps mentioned below. To obtain a working Linux operating system and understand more about the BeagleBone we recommend using the [beagleboard.org](http://www.beagleboard.org) documentation, which can be found at:  
<http://www.beagleboard.org/getting-started>

If a BeagleBone Black is used, then you must boot from a SD card instead of the eMMC. This is because several of the lines used by the eMMC are also used by the RedCape, and it will not be possible to boot into an operating system stored in the eMMC with the RedCape connected. To obtain a bootable SD card for the BeagleBone Black, follow the instructions posted at:  
[http://www.elinux.org/Beagleboard:Updating\\_The\\_Software#Image\\_For\\_Booting\\_From\\_microSD](http://www.elinux.org/Beagleboard:Updating_The_Software#Image_For_Booting_From_microSD)

Once Linux is loaded on the SD card, plug the RedCape on top of the BeagleBone. The result will look like shown in figure 1.3. Then proceed by powering up the BeagleBone and it should boot normally. At this point the RedCape will not power up, this is because it uses a Power Enable signal to activate the internal voltage regulators on the board.

**NOTE:** to use the RedCape it is required to power the BeagleBone from the 5V connector, the USB connection can't supply enough current to power the RedCape (approx. 750mA).



*Figure 1.3 – RedCape mounted on the Beaglebone*

Once the BeagleBone is running and with the RedCape plugged in, connect to the Beaglebone and clone our GIT repository. For this manual we will clone it in the home repository of the BeagleBone, but this is optional (if the cloning directory is different than /home/root, some of the scripts will have to be modified). To clone the GIT repository :

```
root@beaglebone:~# git clone https://github.com/PentaBee/RedCape
```

After that we need to run the script turnON\_RCA1.sh . This will set the output at GPIO1\_16 logic high, which corresponds to setting it to 3.3 V. This action will enable the power supplies of the RedCape.

```
root@beaglebone:~# ./turnON_RCA1.sh
```

Switching the power supplies of the RedCape is achieved by running the script turnOFF\_RCA1.sh. This way you can reset the RedCape at any moment, please bear in mind that by doing this the Cyclone V will lose its configuration and must be reconfigured after power up.

For an automatic power up of the RedCape after a BeagleBone start up, please use the Automatic\_PowerOn.sh script. The script will use Cron to start the RedCape at every start up of the BeagleBone. For this just enter :

```
root@beaglebone:~# ./ Automatic_PowerOn.sh
```

**Note :** After this, each time the BeagleBone starts the GPIO1\_16 will be set as an output and will be set to logic high, the device tree of the RedCape will be loaded as well.

With the RedCape on, use the next command

```
root@beaglebone:~# ./fast_demo.sh
```

The user LEDS on the RedCape should be flashing, congratulations! Your RedCape is now working.

## *1.5 - Stand Alone Operation*

The RedCape can work as an standalone device, by doing this the processing capabilities of the BeagleBone are lost, but the Cyclone V can work as the main

processing unit. The Cyclone V can be configured in standalone usage with the USB Blaster from Terasic, using the JTAG connector in the RedCape (P5). Other option is to use the auto configuration of the Cyclone V using the Flash Memory of the RedCape. For more information in any of those two methods read Chapter 3, configuration options for Cyclone V.

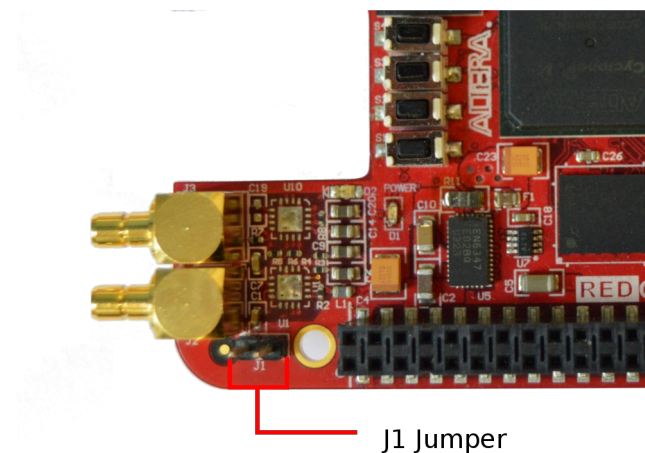
To enable the RedCape standalone operation, is necessary to place a Jumper in the J1 connector of the board. This will hard-wire the Power Enable signal to 5 Volts. After the jumper is set, its necessary to connect 5 volts and ground to the board. Ground should be connected in any of the following pairs of pins :

P1 on RedCape : 1 - 2 , 43 - 44, 45 - 46

P8 on RedCape : 1 - 2

To connect the 5 Volts the pair of pins to use is :

P1 on RedCape : 5 - 6



*Figure 1.4 – J1 jumper for Stand Alone Operation*

**Note :** Do NOT leave the jumper in J1 connected if the RedCape is going to be mounted on the BeagleBone. This will generate the BeagleBone not to boot.

## 2 - Sensors and I<sup>2</sup>C devices

### 2.1 Sensors

The available sensors in the RedCape are:

- 3 Axis Accelerometer (LIS331)
- 3 Axis Gyroscope (L3GD20)
- Integrated IMU (MPU6000)
- Differential Pressure Sensor (MS5611)
- 3D Magnetometer (HMC5883)

All the sensors are connected in parallel and each one has a different I<sup>2</sup>C address, so its possible to specify to which sensor the processor is talking to by using this address. Figure 2.1 shows the available I<sup>2</sup>C devices and their address.

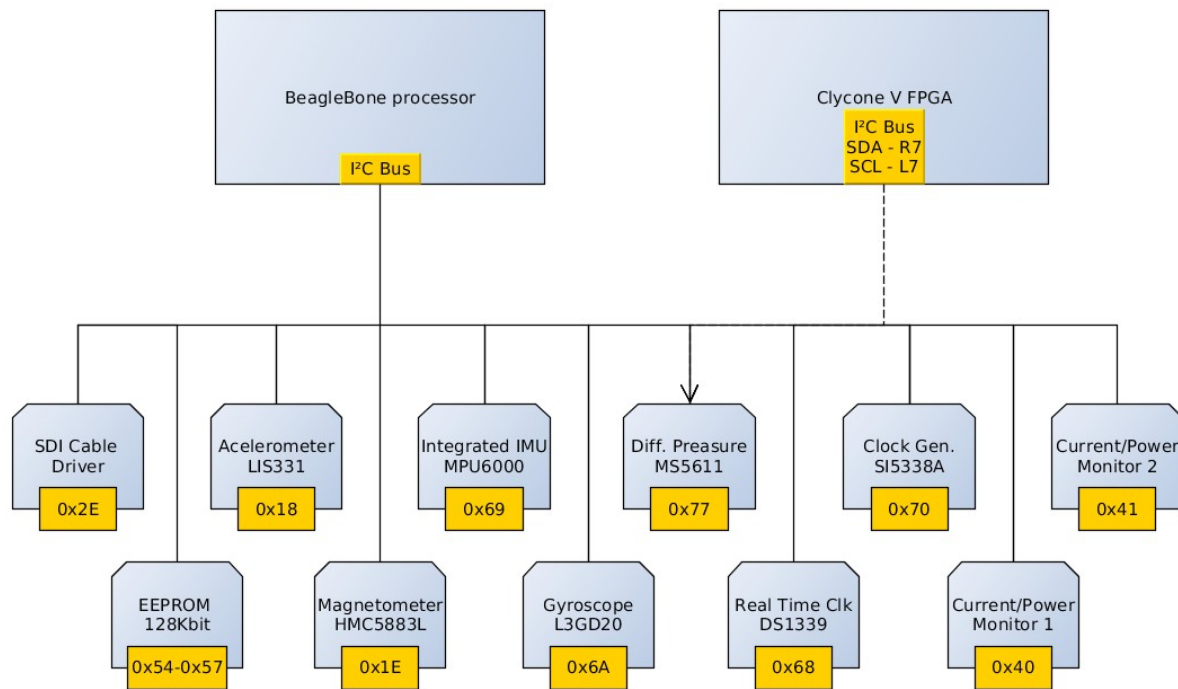


Figure 2.1 – Block diagram of the I<sup>2</sup>C devices

## 2.2 Other I2C devices

### 2.2.1 - ID EEPROM

There is a 128Kbit EEPROM memory available in the RedCape. The intention of this memory device is to store information about the board (Board ID). The data stored contains information about the manufacturer , version number, components, etc. This information is used for loading the necessary device tree overlay, start up scripts and other functions.

The I<sup>2</sup>C address of the EEPROM memory is 0x54 to 0x57. The default address is 0x57, but this address can be changed if it is required. If more cape will be used simultaneously on the BeagleBone, it is required to have a different I<sup>2</sup>C address for the Board ID EEPROM of each cape. To modify the address of the EEPROM please set the Dip Switch 2 in the bottom of the cape. The next table shows the possible values and addresses:

DipSwitch 2 Values		ID EEPROM
D_SW2_1	D_SW2_2	Address
OFF	OFF	0x57
OFF	ON	0x55
ON	OFF	0x56
ON	ON	0x54

It is possible to see the information stored in the EEPROM by using the next command in the beaglebone :

```
root@beaglebone:~# cat /sys/bus/i2c/devices/1-0057/eeprom | hexdump -C
```

The EEPROM has a write protect input, which is connected to the Dip Switch 1 on the top part of the board, Figure 3.1 .To write to the device ensure the switch 3 is OFF.

**NOTE :** The information in this EEPROM may be required by some of the scripts and applications in the BeagleBone. Modify this data **ONLY** if you know what you are doing.

### 2.2.2 - Power Monitor

Two Power monitor chips are available in the RedCape. The used chips are INA219, which can give direct readouts of power monitor, current and voltage. The Power Monitor 1 (which I<sup>2</sup>C address is 0x40) is connected to 1.2 V power supply allowing to read the power consumption from devices using this voltage. The Power Monitor 2(which I<sup>2</sup>C address is 0x41) is connected to 3.3 V.

To use these devices is necessary to configure internal registers. The best way to obtain the configuration values is to use the software created by Texas instruments. To obtain the software and more information on how to use the Power Monitors, refer to : <http://www.ti.com/tool/ina219evm>

### 2.2.3 - Real Time Clock

The DS1339 is a serial real time clock device with a date option, two programmable alarms and a programmable square-wave output. The RedCape has a battery that allows the DS1339 to keep track of the time even when the board is powered off.

**NOTE :** When the RedCape is powered off, it is not possible to read or write I<sup>2</sup>C commands to the real time clock, the battery is used only to keep track of time and does not enable other functions.



## 2.2.4 - Quad Clock Generator

The RedCape has a Si5330 Integrated circuit, which is capable of generating up to 4 differential clock outputs, where each one is independently programmable to any frequency up to 350 MHz. The main goal of this device in the RedCape is to generate clocks for the Cyclone V, but the generated signals can be used for other things. It is possible to use the FPGA as a bridge and send the clock signals to general I/O pins of the FPGA. For more information in how to use this device read section 3.4.2 Quad Clock Generator.

## 2.3 I2C bus connection

The RedCape has a variety of sensors that can be accessed by the BeagleBone processor using I<sup>2</sup>C bus 2, which can be found on pins 19 and 20 of the P1 connector on the RedCape.

It is possible to access these lines from the Cyclone V FPGA as well, so the sensors can be read and controlled directly from the Cyclone V.

**Note:** By default, the Cyclone V is NOT connected to the I<sup>2</sup>C lines, but it can be connected by populating 2 resistors of 0R, in the space for R55 and R56 (bottom of the RedCape).

Inserting R55 and R56 connects the

I<sup>2</sup>C lines to the FPGA I/O-pins:

I <sup>2</sup> C lines		FPGA Pin
SDA	→	R7
SCL	→	L7

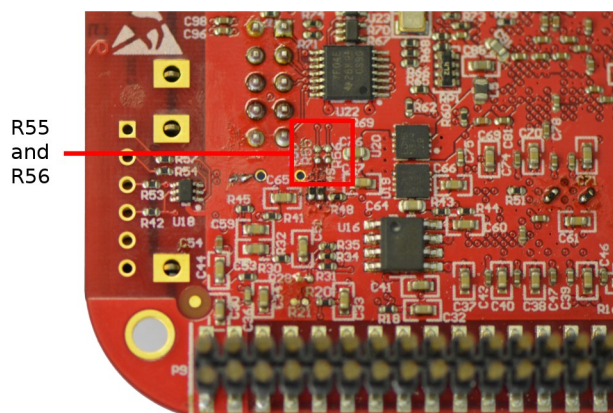


Figure 2.2 – Resistors R55 and R56 for I<sup>2</sup>C with the Cyclone V



An easy way to see the sensors from the Linux environment is to use `i2c-tools`. We have to read the devices on bus 2, to which all the sensors are connected. It's possible to connect more I<sup>2</sup>C devices to the RedCape, as long as they have an address that is not used by other devices. Using the `i2cdetect` should look like this:

```
root@beaglebone:~# i2cdetect -y -r 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  18  --  --  --  --  --  1e  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  40  41  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  UU  UU  UU  UU  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  68  69  6a  --  --  --  --
70:  70  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

Figure 2.3 – Result of `i2cdetect` command

**NOTE:** not all I<sup>2</sup>C devices are detected by `i2cdetect`. For example, the MS5611 pressure sensor on address 0x76 isn't shown in figure 2.3.

From the BeagleBone processor it possible to communicate with the sensors from several different programming languages. A simple library for each sensor has been created, just to initialize and read the raw values of the sensors. The libraries are in C and can be called from other programs. An example of how to use the libraries can be found. For more complex interactions with the sensors is optimal to read the devices data sheets and write the necessary programs.

## 3 - Cyclone V

An FPGA is an electronic device in which any digital circuit can be implemented. The FPGA is a set of arrays of millions of logic programmable cells. One of its main advantages is that different circuits can be implemented in parallel, allowing for a faster processing. Using the FPGA it is possible to implement faster algorithms than in the ARM processor.

The RedCape has an Altera's Cyclone V FPGA as main processing unit. The majority of the components of the RedCape are designed to interact with the Cyclone V, adding with this many possibilities to an already powerful device.

The Altera Cyclone V present in the RedCape is the 5CGXFC3B6U19, this device has 208 general Input/Output pins and 3 XCVR. There are memory blocks of 10Kbits with soft error correction code available. The Cyclone V has the possibility of using embedded hard IP blocks for memory controller, DSP and embedded transceiver I/O. The GX family of the Cyclone V has 31.5 K logic elements, 47600 registers and 3 Gbps transceivers.

The connections for the Cyclone V to the rest of the board will be divided into Configuration Options, Debugging, Peripherals, Sensors and other devices, Connection to the BeagleBone and Memory.

### 3.1 - Configuration Options

Before the FPGA can be used it must be configured. Every time power to the FPGA is interrupted the configuration is lost because it is stored internally in volatile memory. The FPGA configuration data (or programming file) is created by

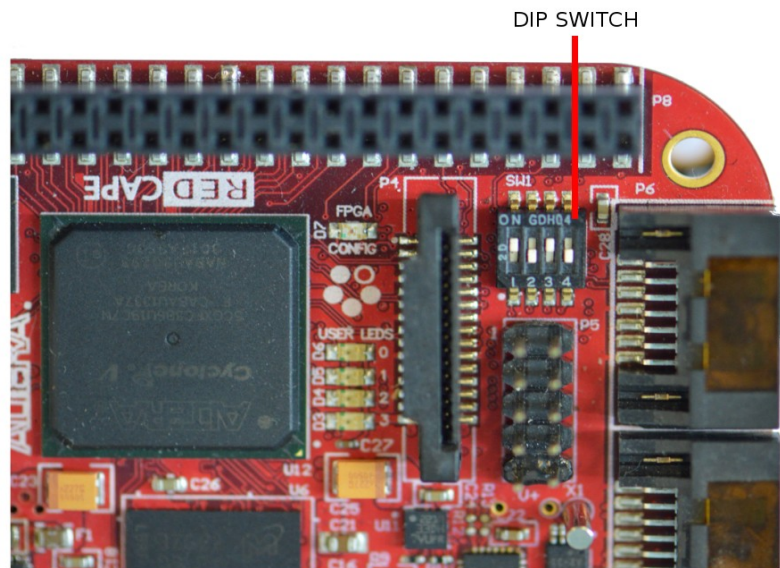
compiling your digital design using dedicated software. The software that this guide will use as a Reference is the Altera Quartus II software. This software can be used for free or bought to obtain full functionalities.  
<http://dl.altera.com/?edition=web>

To explain the full functionality and use of Quartus is out of the scope of this Manual, but full documentation and examples can be found in the Altera website.

Uniquely the Cyclone on the RedCape can be configured using several different methods giving the user a very flexible system whose functions can be changed quickly, a full advantage of combining an embedded system with an FPGA.

The connection for configuration of the Cyclone V to the rest of the board is controlled by the MSEL dip switch, which can be seen in figure 3.1 . The configuration options available through the dip switch selector are as follows :

	MSEL Configuration				Operation
	1	2	3	4	
PIN	0	0	X	X	Fast Passive Serial Configuration for Cyclone V
	1	0	X	X	Standard Passive Serial Configuration for Cyclone V
	0	1	X	X	Fast Active Serial Configuration for Cyclone V
	1	1	X	X	Standard Active Serial Configuration for Cyclone V
	X	X	OFF	X	Write Data to EEPROM (U17) Disable
	X	X	ON	X	Write Data to EEPROM (U17) Enable
	X	X	X	OFF	Cyclone V programming from BeagleBoard is Enabled
	X	X	X	ON	Cyclone V programming from BeagleBoard is Disabled



*Figure 3.1 – Dip Switch in the RedCape*

### 3.1.1 - USB Blaster

The USB Blaster is an external hardware to communicate from a Host PC to the Cyclone V. There are different manufacturers for this device, like Altera and Terasic, and the price is around 40£. Using the USB Blaster allows to debug hardware description code in the FPGA. Using the Quartus software from Altera, and applying the SignalTap II feature its possible to display any program variable in real time.

To use the USB Blaster, connect it in the P5 connector, making sure that the red line in the cable is in the pin labelled as 1 in the P5. Enter Altera Quartus, select Tools → programmer and configure the FPGA.



*Figure 3.2 – USB Blaster connected to the JTAG Port in the RedCape*

### 3.1.1 - Jrunner

One of the possibilities for configuring the Altera Cyclone V FPGA , is to use Jrunner. Jrunner is a C program that runs on the BeagleBone processor and uses four lines to communicate with the JTAG interface in the Cyclone V. Jrunner is a software designed by Altera, and the original source code can be found at:  
<https://www.altera.com/download/legacy/jrunner/dnl-jrunner.html>

The Jrunner version for the RedCape is based on the Alter code but modified to work in the BeagleBone Black and to use the correct pin assignments for the Cyclone V . The source code used can be obtained from the /src directory of the Jrunner folder.

Below we outline hoe to use Jrunner for the RedCape:

#### Step 1 – Obtain Configuring files

Jrunner allows the user to configure the FPGA using the embedded processor, but to do this its necessary to have a working project configuration for the FPGA. The project must be saved as an Raw Binary File (RBF) file to be sent to the FPGA. To obtain the RBF file its possible to use Altera Quartus. From the Quartus main window, with the selected project open, follow the next steps:

File → Convert programming Files

In the Convert programming File window, select the next

Programming file type : Raw Binary File (RBF)

Mode : 1 bit passive serial

Enter the desired output file name. In desired Files to convert, follow the next steps :

Select : SOF DATA and Add File ( Here its necessary to select the .sof file of the compiled project, its usually in the /output\_files directory)

Then just click Generate, the RBF file should be ready.

Now we need to obtain an .cdf file that will tell Jrunner some information about the FPGA and the RBF. The easiest way to do this is to open the “cog\_profiler.cdf” file and modify the name file accordingly. The file contains the next text :

```
/* Quartus II Version 9.1 Build 350 03/24/2010 Service Pack 2 SJ Full Version */
```

```
JedecChain;
```

```
FileRevision(JESD32A);
```

```
DefaultMfr(6E);
```

```
P ActionCode(Cfg)
```

```
Device PartName(5CGXFC3B6U19) Path("") File("file.rbf") MfrSpec(OpMask(1));
```

```
ChainEnd;
```

```
AlteraBegin;
```

```
ChainType(JTAG);
```

```
AlteraEnd;
```

It is necessary to modify the File value, so change “file.rbf” for the name given to the RBF in the Altera Convert Programming File. Its maybe simpler to have a .cdf file for each RBF file, so its a good idea to save the .cdf file with the same name as the .rbf file to which is pointing.

**NOTE : The RBF file must have its name all in caps, both in the directory and in the reference to it in the .cdf file.**

Once all these files are ready, it is necessary to put all of them (.rbf, .cdf, jrunner) together in the same directory.

## Step 2 – Setting the DipSwitch

Set the correct mode in the DipSwitch. The connection of the BeagleBone processor to the FPGA for the JTAG configuration is controlled by a 4-bit bidirectional translator, so it needed to activate the desired mode. For this, the bit 4 of the DipSwitch (SW1) in the RedCape is used. If the bit is OFF then the communication is possible. So ensure to set this bit to OFF before using the Jrunner. If the bit is set to ON then the jrunner will show a “ JTAG chain broken! “ Error.

## Step 3 - Calling jrunner from command line.

After obtaining the files from the Github Repository, its necessary to give the right permissions to the jrunner file. To do so type :

```
root@beaglebone:~/Jrunner# chmod +x jrunner
```

With the right permissions we can now execute the file. So the usage for configuring the Cyclone V FPGA is :

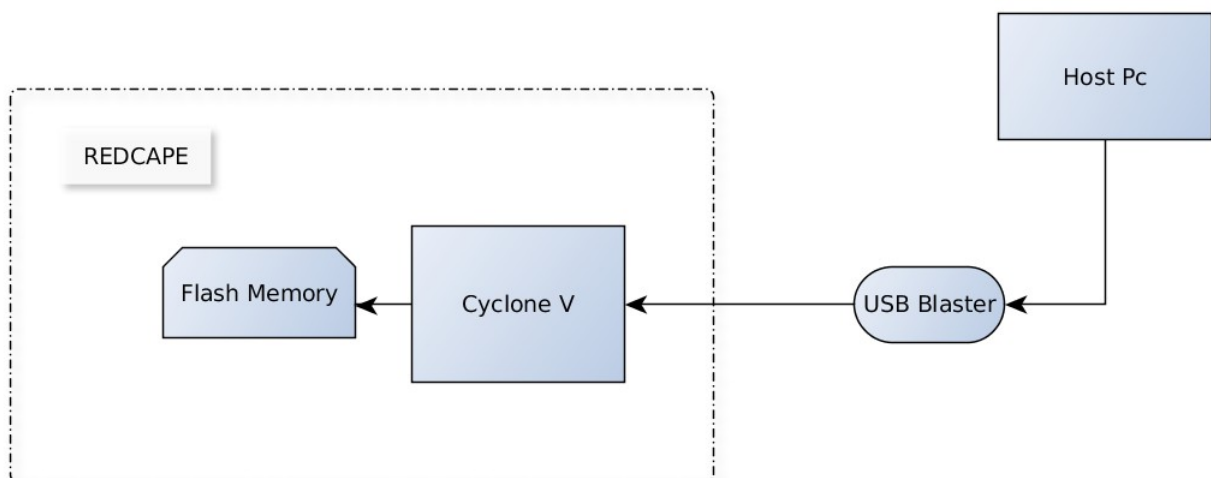
```
root@beaglebone:~/Jrunner# ./jrunner redcapea0_demo.cdf
```

to configure the demo project. For another project just replace the name of the .cdf file.

### 3.1.3 - Active Serial with USB Blaster

One of the options to configure the Cyclone V is Active serial configuration in which the Cyclone V will be the master and ask for the configuration data to a flash memory. The RedCapE has a Serial NOR Flash Memory of 64 Mb (N25Q064) available for this. So at power up, the Cyclone V will ask for the configuration to the flash memory, starting to configure the device intermediately after start up. This is a good procedure if one configuration has to be used repeatedly.

For active serial configuration is necessary to add the project data to the Flash memory so it can give the data to the Cyclone V later on. On the RedCapE there is no direct link to the Flash memory, the only connection to it is with the Cyclone V. One must therefore use the Cyclone V as a bridge to pass the communication from the USB Blaster to the Flash memory.



*Figure 3.3 – Block Diagram of Flash Memory Programming for Active Serial Configuration*

To pass the configuration data to the Flash memory:

#### Step 1 – Convert .SOF file to . JIC.

File → Convert programming Files



In the Convert programming File window, select the next  
 Programming file type : JTAG indirect configuration file (JIC)  
 Mode : active serial  
 Configuration Device : EPCQ64  
 Advance → Disable Id Check

Enter the desired output file name. In desired Files to convert, follow the next steps :

Select Flash Loader : Add Device → Cyclone V - 5CGXFC3B6

Select : SOF DATA and Add File ( Here its necessary to select the .sof file of the compiled project, its usually in the /output\_files directory)

Then just click Generate, the JIC file should be ready.

## Step 2 - Setting the Dip Switch

Set the correct mode in the Dip Switch. To enable Active Serial configuration the bit 1 of the Dip Switch (SW1) should be on ON and the bit 2 should be on OFF. The state of the other two bits make no difference for active serial configuration.

## Step 3 – Program the Flash memory

Connect the USB Blaster like for the USB Blaster configuration example.

Open Tools → Programmer, and click on Add File, add the .jic file and select both program and configure. Start the process.

After this, the Flash memory should have the necessary data to configure the Cyclone V. Ensure the Dip switch is still in the right values for Active serial configuration and turn off and on the RedCape.

```
root@beaglebone:~# ./turnOFF_RCA1.sh
root@beaglebone:~# ./turnON_RCA1.sh
```

The Cyclone V should configure itself intermediately after power on

### 3.1.4 - Active Serial with Jam player

Other option to use active serial configuration is to use a software called Jam player. This software is able to save the configuration data into the Flash memory, using the Cyclone V as a bridge, from the BeagleBone processor. By using this modality, it is not necessary to use an USB Blaster. To do this, follow the next steps:

## Step 1 – Obtain a JIC File

Like in the steps for Active serial with USB Blaster, we need to obtain a JIC file, so follow that step.



## Step 2- Obtain a .JAM file

Tools → Programmer and click on Add File, add the .jic file.

File → Create JAM

Save the Jam file and move it to the JamPlayer directory in the BeagleBone

## Step 3 – Use Jamplayer

In the BeagleBone enter the jamplayer directory :

```
root@beaglebone:~/# cd jamplayer/
```

Use the set script to add the needed configuration to the GPIO pins required

```
root@beaglebone:~/jamplayer/# ./set.sh
```

Configure and Program the Flash player

```
root@beaglebone:~/jamplayer/# ./jp -aconfigure X.jam && ./jp -aprogram X.jam
```

Modify the "X.jam" for the actual name of your jam file. The process will take around 2 minutes and then the Flash memory will be programmed.

**NOTE :** Remember to have the DipSwitch set for Active serial Communication as seen in the Active Serial Communication with USB Blaster Steps.

## 3.2 - Debugging

For easier debugging and visualisation of the state of the Cyclone V in the RedCape, there are 5 LEDs controlled by the Cyclone V and 4 Push Button that as inputs.

### 3.2.1 - LEDS

There are 5 available LEDs in the RedCape for the Cyclone V, from this 4 are for user configuration and 1 is for visualisation of the state of the FPGA. The Config FPGA LED (D7) is a yellow LED that will be active only when the Cyclone V has been configured, the rest of the time the LED will be off. This LED can not be directly controlled by the user.

## User LEDs

The RedCape have 4 user LEDs, which are connected to outputs of the Cyclone V. The LEDs are connected in inverted logic, meaning that when the output in the Cyclone V is in low, the LED will be turned on, and vice versa. The connections for the user LEDs [0 – 3] to the Cyclone V are as follow:

User LED	Cyclone V Pin
0	T9
1	P7
2	T7
3	T8

### 3.2.2 - Push Button

The RedCape have 4 push buttons as inputs for the Cyclone V. The buttons are in negated logic, meaning that when the button is not pushed, the Cyclone V will read a digital HIGH and when the button is pushed the reading will be a digital LOW. The connections of the buttons are as follow :

Push Button	Cyclone V Pin
S1	T20
S2	T19
S3	R22
S4	T22

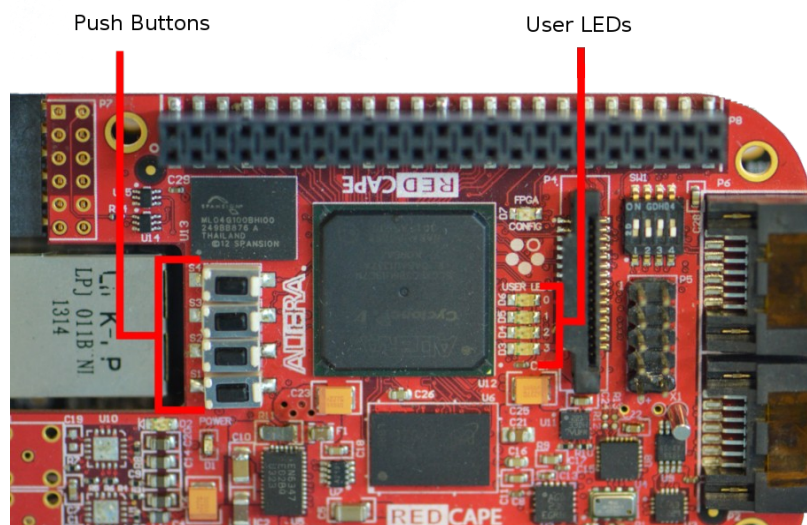


Figure 3.4 – LEDs and Buttons of the RedCape

### 3.2.3 - Clock

The RedCape has a 25MHz oscillator that can be used as a clock input in the FPGA for any synchronous activity. The oscillator is connected to the Pin H10 and its directly power, so there is no need to activate it. If more clocks are desired or of different frequencies, read the section of the Quad Clock Generator in the Sensors and Other Devices section.

## 3.3 - *Peripherals*

The RedCape has connectors designated for special peripherals, these connections are made to the Cyclone V general input/output pins. This means that the connections can be used for the designed peripherals or for any other application or boards developed by the user. In this section we describe the connectors available, the peripherals for which they were designed, and the schematics of the connector so the user can use it for a different application if needed.

**NOTE :** The FPGA I/O pin do not have any EMC protection, so its important that the user be careful when making connections.

### 3.3.1 - JTAG connector

The JTAG connector is the P5 in the RedCape, its a 10-pin connector and its designed to be used with the USB Blaster. This connector allows the external hardware to communicate to the signals TDO, TMS, TCK, TDI from the Cyclone V. These pins are not general I/O-pins, but form a dedicated programming and debugging interface. Using the Signal Tap feature of Quartus II a user has a dedicated debugging tools that can record internal signals making a logic analyser a thing of the past.

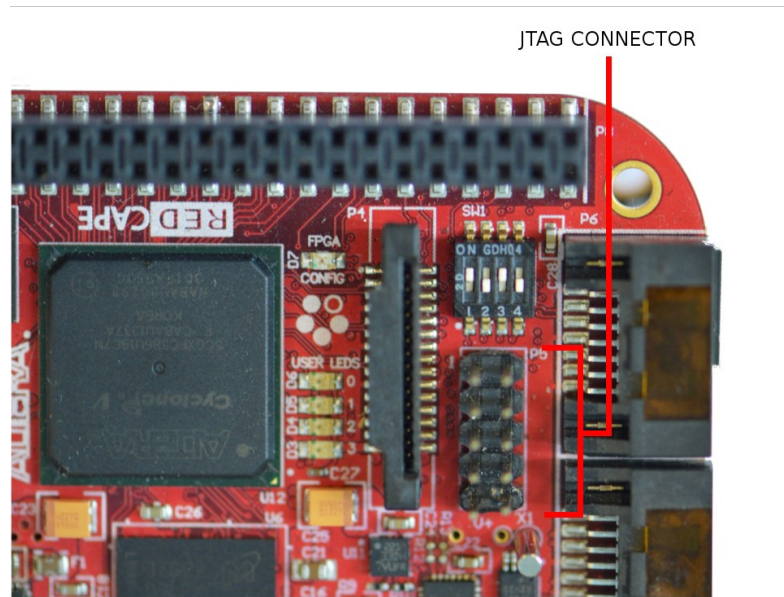


Figure 3.5 – JTAG Connector on the RedCape

### 3.3.2 - PFC -15 Connector (RaspberryPi cam)

This connector is for a 15-way flex cable. It is commonly used for the RaspberryPi camera module, but it also has other applications. In the RedCape this connector is labelled as P4, on the top side of the board.

Its possible to connect a Raspberry Cam and obtain the data to the Cyclone V, providing this way the possibility of parallel process of the image data. The camera sensor can be controlled with I<sup>2</sup>C commands, and it is connected to the I<sup>2</sup>C lines of the BeagleBone processor. The connections for this connector are as follows :

PFC – 15 Connector	Cyclone V Ball	External
1		GND
2	AA1	
3	AA2	
4		GND
5	W1	
6	W2	
7		GND
8	R1	
9	R2	
10		GND
11		
12		
13		I <sup>2</sup> C2 SCL
14		I <sup>2</sup> C2 SDA
15		VCC (3.3V)

Table of connections in connector PFC -15 ( P4 )

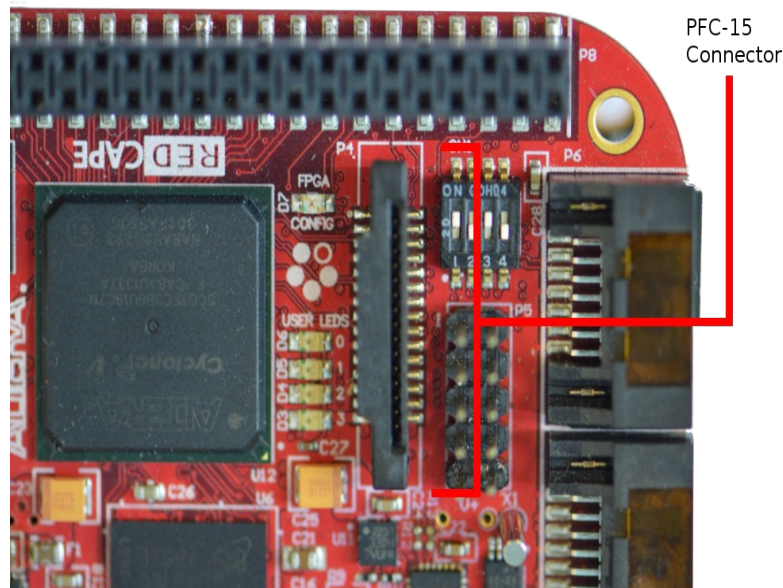


Figure 3.6 – PFC -15 connector for Raspberry Pi Cam

**NOTE:** The Cyclone V I/O-pins for the PFC-15 connector are the same used for SATA 0, SATA 1 and SDI. So it is not possible to use this connector as the same time as any of those others.

### 3.3.3 -PMODS

PMOD is an interface specification by Diligent , used to connect low frequency, low I/O pin count peripheral modules to host controller boards. In this case, the RedCape is the host controller board ,and the connections in the Pmod are a bridge to the Cyclone V general input output/pins. There are six pins and twelve pins versions of this connectors, and the RedCape has them both. The 6-pin connector is in P3, right under the SATA connector, and the twelve pin connector is in P7.

Pmod -12 Connector

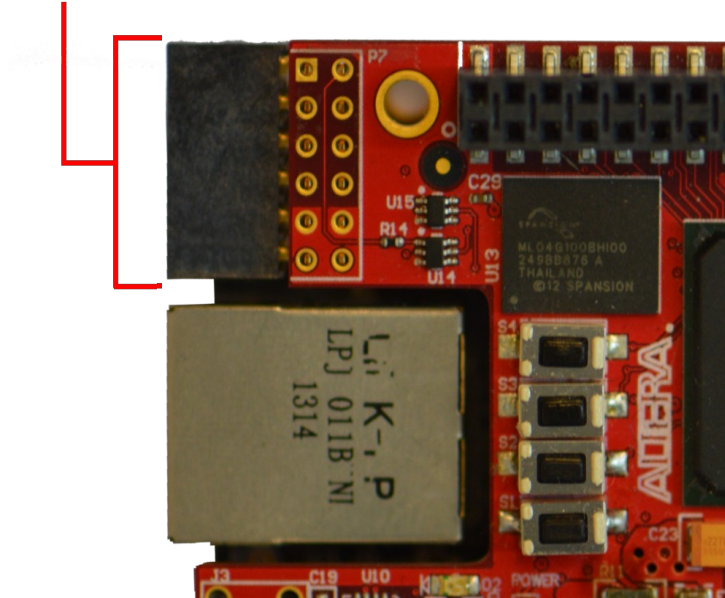


Figure 3.7 – Pmod -12 Connector

There is a wide range of possible peripheral boards available for these connectors, made by Diligent. These boards have functionalities such as: Capacitive I/O, Stereo Power Amplifier, GPS Receiver, OLED Graphic Display, Bluetooth Interface, USB to UART interface, just to name a few. All these peripherals can be connected to the Cyclone V using P3 and P7. The Diligent web page provides a lot of VHDL code for PMOD applications. Simply by recompiling the FPGA code for the Cyclone V these applications can be used on the RedCape. For more information on these peripheral boards please check: <http://www.digilentinc.com/Products/>

It is also possible to connect any board designed by the user. To do this, it is necessary to know how the connections are made. Please refer to the Schematics of the RedCape in the Appendixes or use the connection table shown below.

Pmod -12	Cyclone V Ball	External
1	T17	
2	T18	
3	R16	
4	R17	
5		GND
6		VCC (3.3V)
7	R20	
8	R19	
9	R21	
10	P19	
11		GND
12		VCC (3.3V)



Pmod -6 (P3)	Cyclone V Ball	External
1	E20	
2	F20	
3	H20	
4	H19	
5		GND
6		VCC (3.3V)

### 3.3.4 - SATA connectors

The RedCape has two SATA connectors (P2 and P6) that can be used to connect a Hard Drive (or other device using SATA connectors). A SATA interface is used to connect mass storage devices such as optical drives and hard disks. The SATA connectors in the RedCape are a bridge to the transceiver pins of the Cyclone V. To connect a mass storage device is necessary to generate a interface protocol design for the Cyclone V.

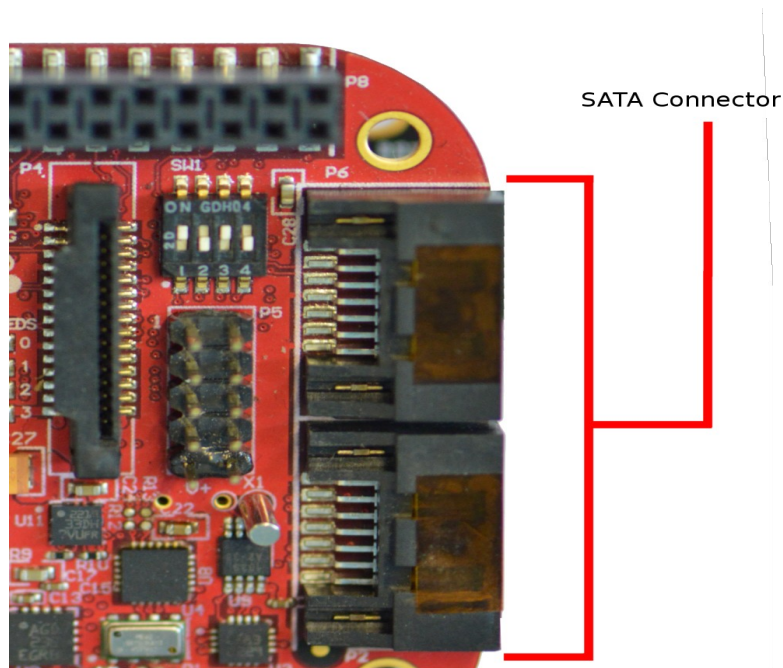


Figure 3.8 – SATA connectors in the RedCape

To make the connections of the Cyclone V to the SATA interface use the next table.

SATA_0	Cyclone V Ball	External
1		GND
2	Y4	
3	Y3	
4		GND
5	AA2	
6	AA1	
7		GND
8		CH_GND

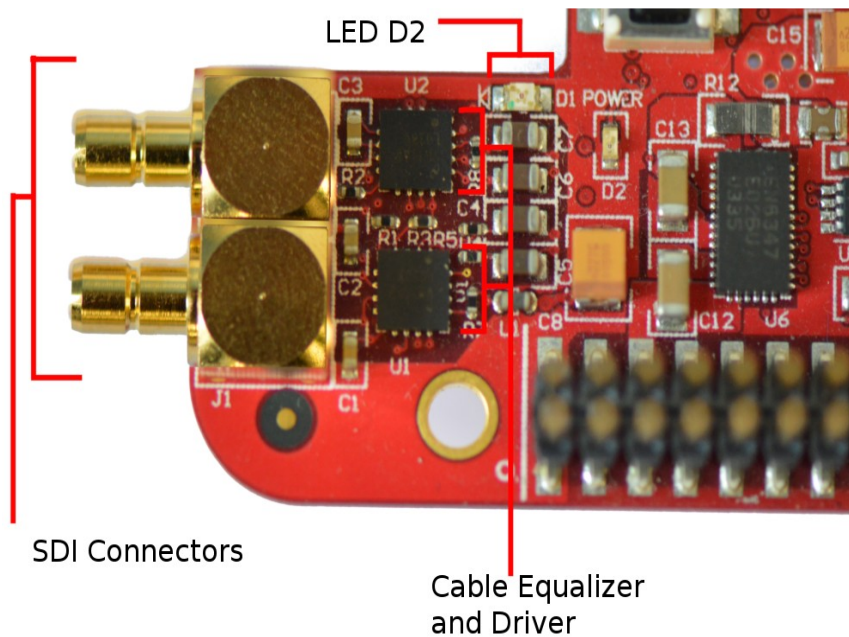
SATA_1	Cyclone V Ball	External
1		GND
2	U2	
3	U1	
4		GND
5	W2	
6	W1	
7		GND
8		CH_GND

### 3.3.5 - SDI Video

The RedCape has the ability to interface to Serial Digital Interface Video (SDI) devices such as cameras. This type of video interface uses BNC connectors. There are two connections of SDI video in the RedCape (J2 and J3), which can be used one as an input connection and the other as an output, providing the ability to process digital video in real time. For this function, the RedCape uses an SDI Cable Equalizer (LMH0384SQ) for the SDI video input and a SDI Cable Driver (LMH0303Q) for the output.

When the SDI video input is activated, the LED D2 powers up. Both SDI drivers are connected to the FPGA, so the video data can enter and be processed by the Cyclone V. It is possible to control the output cable driver by the I<sup>2</sup>C lines as well, either from the BeagleBone processor or from the Cyclone V. To interact with SDI video please refer to the schematics and the datasheets of the drivers.





*Figure 3.9 - SDI connectors in the RedCape*

### 3.3.6 - P8 expansion port

To use the rest of the general input/output pins of the Cyclone V, it's possible to use the port P8. Port P8 is connected to 42 general input/output pins, and it can be used in many applications. These pins have an input and output voltage logic of 3.3 V. To see the connections of P8 to the Cyclone V Balls, please refer to the tables of connections in the Appendix.

**NOTE:** The FPGA I/O-lines exposed on P8 do not have any EMC protection, so it is important that the user is careful when making connections.

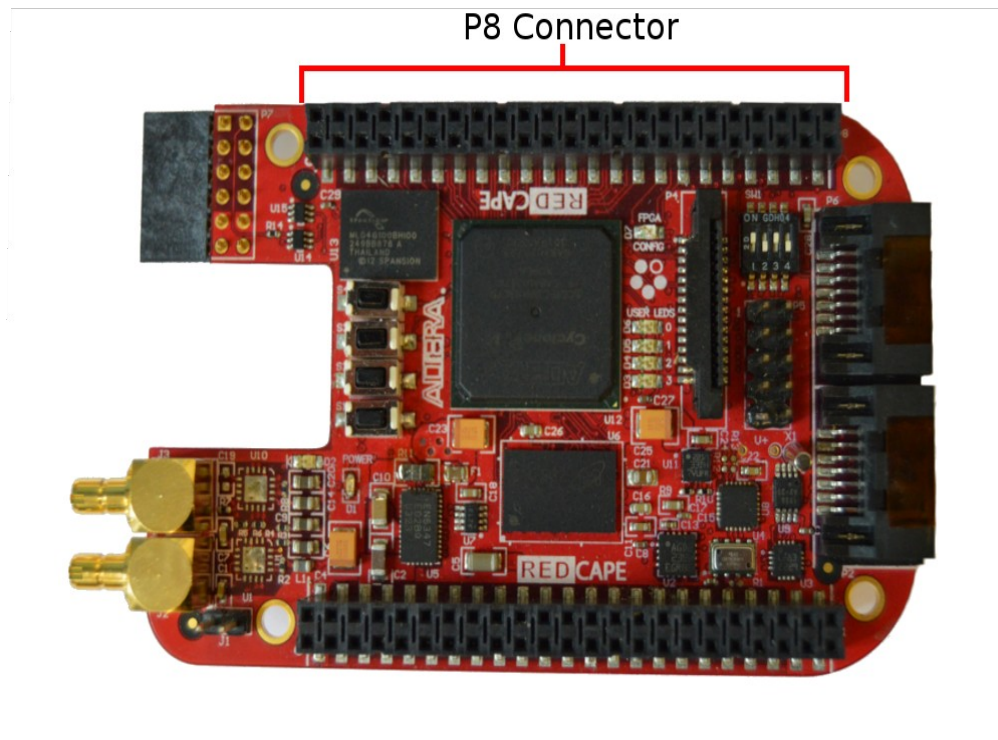


Figure 3.10 – P8 expansion port

## 3.4 - Sensors and Other devices

### 3.4.1 - Sensors

The RedCape has a variety of sensors, these sensors are connected to the I<sup>2</sup>C bus and can be controlled from the Cyclone V. Before you can use the FPGA to act as a master (or slave) on the I<sup>2</sup>C bus you must solder in two links, for more info in this check the Sensors Section.

To control and communicate with the sensors via the I<sup>2</sup>C is necessary to generate an I<sup>2</sup>C protocol driver in the Cyclone V. It is possible to use the I<sup>2</sup>C IP by Altera in Opencores for free. Also it is possible to design your own I<sup>2</sup>C controller. Then the Cyclone V can be used as a Slave or as a Master, depending on the configuration. As a Master it can control and read the sensors, as a Slave it can communicate with the BeagleBone.

### 3.4.2 - Quad Clock Generator

The RedCape has a Si5330 Integrated circuit, which is capable of generating up to 4 differential clock outputs, where each one is independently programmable to any frequency up to 350 MHz. Using this device its possible to have as inputs for the Cyclone V clocks of different frequencies.

To use the Quad Clock Generator, use the next table for pin connections.

Quad Clock Gen	Cyclone V Ball	External
CLK0A	V4	
CLK0B	U4	
CLK1A	L17	
CLK2A	R10	
CLK3A		Feedback clock
CLKIN	T12	

The Quad Clock Generator can also be communicated from the I<sup>2</sup>C interface. Controlling the device is by changing the values of its registers. The amount of registers for this device is large, making by hand configuration tedious. Fortunately there is an application called Clock Builder Desktop. This application (Made for Windows) allows the user to configure the desired behaviour of the Quad Clock generator and obtain a C header with all the values of the registers.

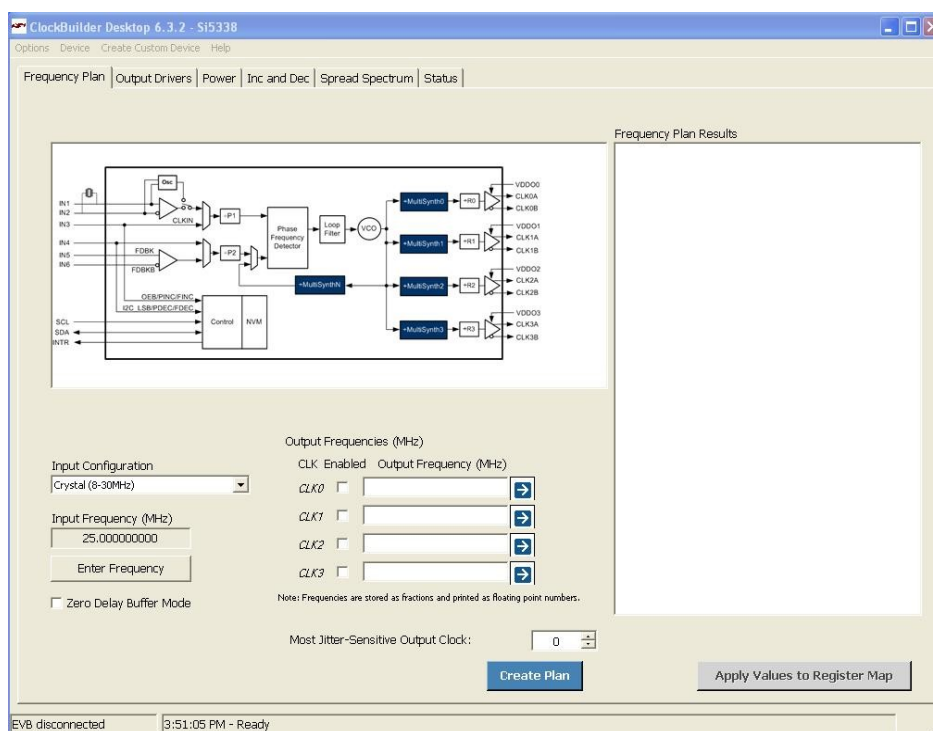


Figure 3.11 – Window of the Clock Builder Desktop

To use the Quad Clock Generator download the Clock Builder Desktop from <http://www.silabs.com/products/clocksoscillators/clock-generator/Pages/clockbuilder-desktop.aspx> . Install, open and follow the instructions until the desired behaviour of the clocks is ready. Then select Options → Save as C code header file .

Save the file with the name register\_map.h . Enter the directory /Sensors\_and\_Drivers /SI5338\_conf/ and substitute the old file for the file that had

been just generated by the Clock Builder Desktop. Compile clock\_conf.c and run it. The Quad Clock Generator should be now working. If you have any doubts read the data sheet for the SI5338 and read the comments on clock\_conf.c .

### 3.5 - Connection to the BeagleBone

To interact all the possibilities of the BeagleBone ARM processor with the Cyclone V in the RedCape, it's necessary to have a good communication protocol between this two. The protocol in use in the RedCape is the General Purpose Memory Controller (GPMC).

The GPMC is a unified memory controller dedicated to interfacing with external memory devices. In this case, we will configure the Cyclone V in such a way that it will behave as a memory device. The GPMC is a module inside the BeagleBone processor, and we need to call it so we can move data fast and without putting to much load into the processor.

To use the GPMC from the BeagleBone processor, its necessary to configure it. Each chip-select behaves independently and its programmable in it's control signal parameters. In the RedCape, the proposed schema requires the Cyclone V to behave as a NOR Flash-like memory. The chip-select used for this is can be the CS\_5 or the CS\_6, both are connected to the Cyclone V.

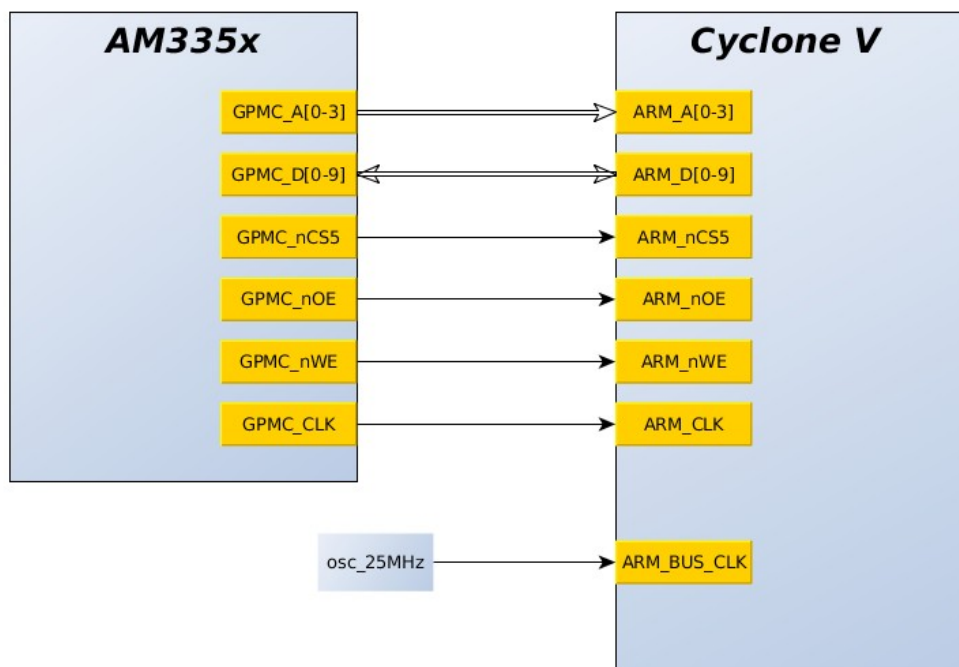


Figure 3.12 – GPMC Block Diagram

For more information on the GPMC read the data sheet for the BeagleBone processor (AM335x), use the table of connections for the GPMC in the Appendixes and follow the Demo of PWM in the Cyclone V controlled by network.

**NOTE:** To be able to use the GPMC make sure that the RedCape Device Tree has been loaded. If it is not loaded the signals will not get to the correct pins and the GPMC communications will not work.

## 3.6 - Memory

If more memory space is required, the RedCape has two devices to store extra data. The RedCape has a DDR3 SDRAM memory and an NAND Flash memory.

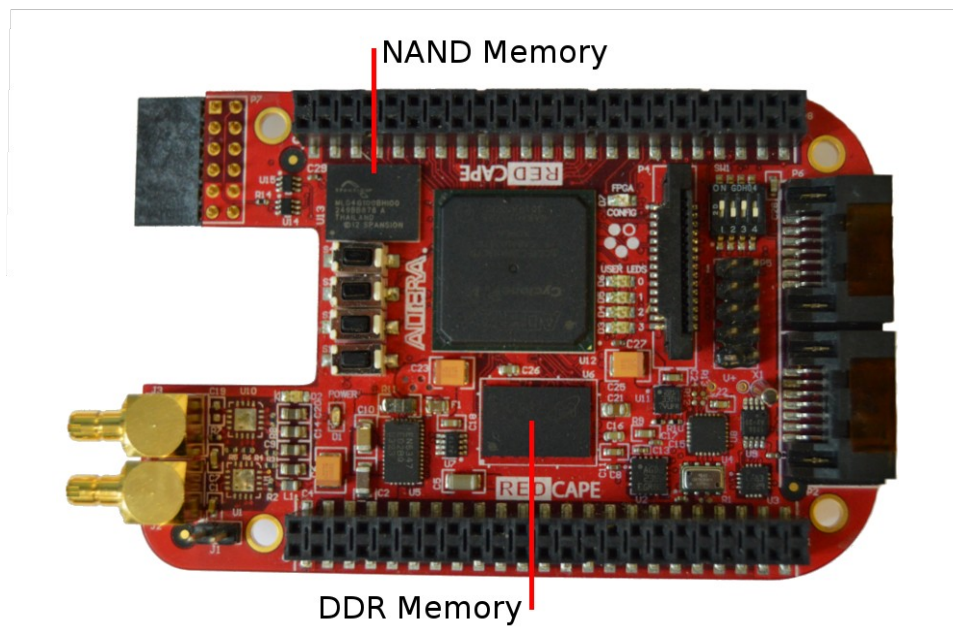
### 3.6.1 - DDR3 SDRAM memory

There is a 4Gb DDR3 SDRAM memory on the RedCape which is interfaced to the Cyclone V. Controlling a DDR3 memory device requires a complex interface. Fortunately Altera provides two solutions: a hard memory controller and soft interface (implemented in VHDL or Verilog). Using the Megawizard feature of Quartus, it is possible to generate code for interfacing to the DDR3 memory. To implement this use the table in Appendixes for connections between the Cyclone V and the DDR3.

### 3.6.2 - NAND Flash memory

There is a 4Gb NAND Flash memory in the RedCape. It is possible to communicate with the NAND memory using the GPMC Bus. As seen before (in the Connection to the BeagleBone section), in this bus is also the Cyclone V and the BeagleBone processor. Using the GPMC module in the BeagleBone processor is possible to access the information in the NAND memory, even to save here an Operative System and boot the BeagleBone from it. Effectively the NAND memory then replaces the eMMC memory chip on the BeagleBone Black.

It is possible to implement a Memory controller for the NAND memory from the Cyclone V. For this is possible to buy or design the controller. Use the connection table in the Appendixes to implement the communication with the NAND memory. The chip-select for the NAND memory is CS0.



*Figure 3.13 – Memories in the RedCape*



## 4 – Code Examples

### 4.1 - Use of the I<sup>2</sup>C devices in the RedCape, accelerometer

For this example we will use one of the I<sup>2</sup>C compatible sensors in the RedCape , specifically we will use the 3 axis accelerometer ( LIS331). The MEMS accelerometer is in the top side of the RedCape, connected to I<sup>2</sup>C\_2 bus, and it has the address 0x18 . To use this sensor we need to modify some of the values in its registers, so it changes from a sleep state to an active state. For more information in the use of this device refer to the data sheet:  
<http://www.st.com/web/en/resource/technical/document/datasheet/CD00213470.pdf> .

For this example we will use the code in the /Demo directory , inside the Git Hub repository. So from the BeagleBone start by checking that the sensor is connected and powered up. For this use the i2cdetect tool :

```
root@beaglebone:~# i2cdetect -y -r 1
```

It should look like this :

```

root@beaglebone:~# i2cdetect -y -r 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  18  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  40  41  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  UU  UU  UU  UU  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  68  69  6a  --  --  --  --
70:  70  --  --  --  --  --  --  --  --  --  --  --  --  --  --

```

Figure 4.1 – results from *i2cdetect* command

Since we can see the address 0x0018 recognized then the sensor is connected and powered up. The next step is to run the C program to see the raw values. For this, enter the /Demo directory, then the /LIS331 and run demo\_lis331. From the command line it should be like this:

```

root@beaglebone:~# cd DEMO
root@beaglebone:~/DEMO# cd sensors
root@beaglebone:~/DEMO/sensors# ./lis331_drv

```

Then it should be visible the output of the sensor. This are the raw values of acceleration given by the sensor. The output should look like this :

```

root@beaglebone:~# cd DEMO/
root@beaglebone:~/DEMO# cd sensors/
root@beaglebone:~/DEMO/sensors# ./lis331_drv

Initialise hardware
The value of the acceleration vector is : 320 X, 416 Y, 400 Z
The value of the acceleration vector is : 256 X, 352 Y, 368 Z
The value of the acceleration vector is : 304 X, 432 Y, 304 Z
The value of the acceleration vector is : 256 X, 368 Y, 400 Z
The value of the acceleration vector is : 192 X, 352 Y, 336 Z
The value of the acceleration vector is : 304 X, 400 Y, 432 Z
The value of the acceleration vector is : 256 X, 384 Y, 336 Z
The value of the acceleration vector is : 288 X, 400 Y, 320 Z
The value of the acceleration vector is : 272 X, 336 Y, 416 Z
The value of the acceleration vector is : 288 X, 432 Y, 416 Z
The value of the acceleration vector is : 320 X, 512 Y, 432 Z

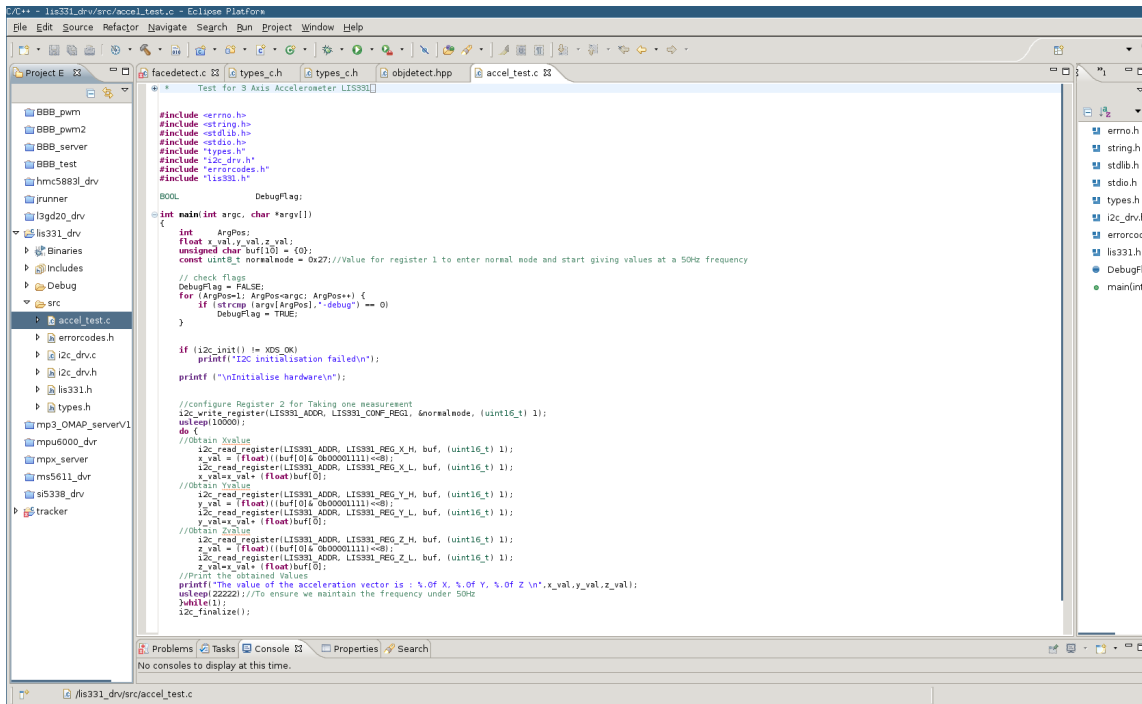
```

Figure 4.2 – Output of demo for LIS331

For modifying the code and obtaining the functionality required from the sensor, enter the /src directory and modify the C code. For doing this we recommend using a cross compilation from a Host PC. For this manual we will use a Linux distribution Host Pc running Ubuntu 12.10 and Eclipse for C programming and compilation. For the cross compilation is necessary to obtain the gcc compiler for the BeagleBone



processor. For more information about cross compilation refer to the BeagleBone Reference Manual. The code can also be modified and compiled from the BeagleBone enviroment itself.



```

#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <types.h>
#include "i2c_drv.h"
#include "errorcodes.h"
#include "lis331.h"

BOOL DebugFlag;

int main(int argc, char *argv[])
{
    int ArgPos;
    float x_val, y_val, z_val;
    unsigned char buf[10] = {0};
    const uint8_t normalmode = 0x27; // Value for register 1 to enter normal mode and start giving values at a 50Hz frequency

    // check flags
    DebugFlag = FALSE;
    for (ArgPos=1; ArgPos<argc; ArgPos++) {
        if (strcmp (argv[ArgPos], "-debug") == 0)
            DebugFlag = TRUE;
    }

    if (i2c_init() != XDS_OK)
        printf("I2C Initialisation failed\n");
    printf("\nInitialise hardware\n");

    //configure Register 2 for Taking one measurement
    I2C_write_register(LIS331_ADDR, LIS331_CONF_REG, &normalmode, (uint16_t) 1);
    usleep(10000);
    do {
        //Obtain X value
        I2C_read_register(LIS331_ADDR, LIS331_REG_X_M, buf, (uint16_t) 1);
        x_val = (float)((buf[0] & 0b00001111) << 8);
        I2C_read_register(LIS331_ADDR, LIS331_REG_X_L, buf, (uint16_t) 1);
        x_val = (float)((x_val << 8) | (float)buf[0]);

        //Obtain Y value
        I2C_read_register(LIS331_ADDR, LIS331_REG_Y_M, buf, (uint16_t) 1);
        y_val = (float)((buf[0] & 0b00001111) << 8);
        I2C_read_register(LIS331_ADDR, LIS331_REG_Y_L, buf, (uint16_t) 1);
        y_val = (float)((y_val << 8) | (float)buf[0]);

        //Obtain Z value
        I2C_read_register(LIS331_ADDR, LIS331_REG_Z_M, buf, (uint16_t) 1);
        z_val = (float)((buf[0] & 0b00001111) << 8);
        I2C_read_register(LIS331_ADDR, LIS331_REG_Z_L, buf, (uint16_t) 1);
        z_val = (float)((z_val << 8) | (float)buf[0]);

        //Print the obtained values
        printf("The value of the acceleration vector is : %f X, %f Y, %f Z \n", x_val, y_val, z_val);
        usleep(22222); //To ensure we maintain the frequency under 50Hz
    } while(1);
    I2C_finalize();
}

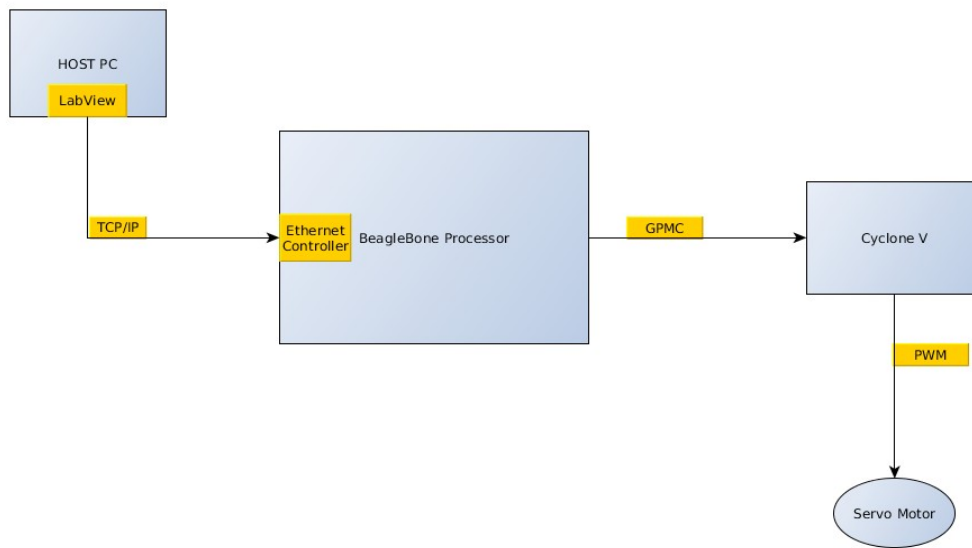
```

Figure 4.3 – Eclipse window with the code for the LIS331 example

## 4.2 - PWM from Cyclone V

In this example we will use the Cyclone V to generate a PWM signal to control a servo motor. The servo motor will be the SAVOX SC- 1258TG Digital Servo. The operation frequency of the PWM for this servo motor is of 50 Hz. To control the PWM output of the Cyclone V, and by that the position fo the servo motor, we will use a GUI in the Host PC and send the data trough TCP/IP l. The GUI for this will be created in Labview.

The work flow is as follows: TheLabview GUI sends the desired data over TCP/IP of the desired position on the servomotor. The BeagleBone receives this data trough its Ethernet controller and sends it to the BeagleBone processor. The processor will obtain this data and send the correct instructions to the Cyclone V through the GPMC bus. The Cyclone V will obtain this instructions and modify its PWM generation module accordingly. The output of the PWM will be sent to the servo motor, controlling its position.



*Figure 4.4 – Block Diagram of PWM control example*

To use this example enter the /DEMO directory, then /PWM . From this point the first thing we need to do is to configure the Cyclone V with the pwm\_c5 project. For this run jrunner and select the pwm\_c5 configuration. That can be done as follows.

```

root@beaglebone:~# cd DEMO
root@beaglebone:~/DEMO# cd PWM
root@beaglebone:~/DEMO/PWM# ./jrunner pwm_c5.cdf
  
```

The PWM outputs are set to 3 of the user LEDs by the moment, in order to have a visible display of the PWM working. If it is desired to connect to a servomotor to control it, then it is compulsory to change the GPIO to which the outputs are connected. To do this, inside quartus with the project open, select Assignments -> Pin Planner. In Pin planner look for the values of pwm to pwm2 and modified them to the desired ones. Use the table "Connectors of FPGA and P8" to know which Ball to use.

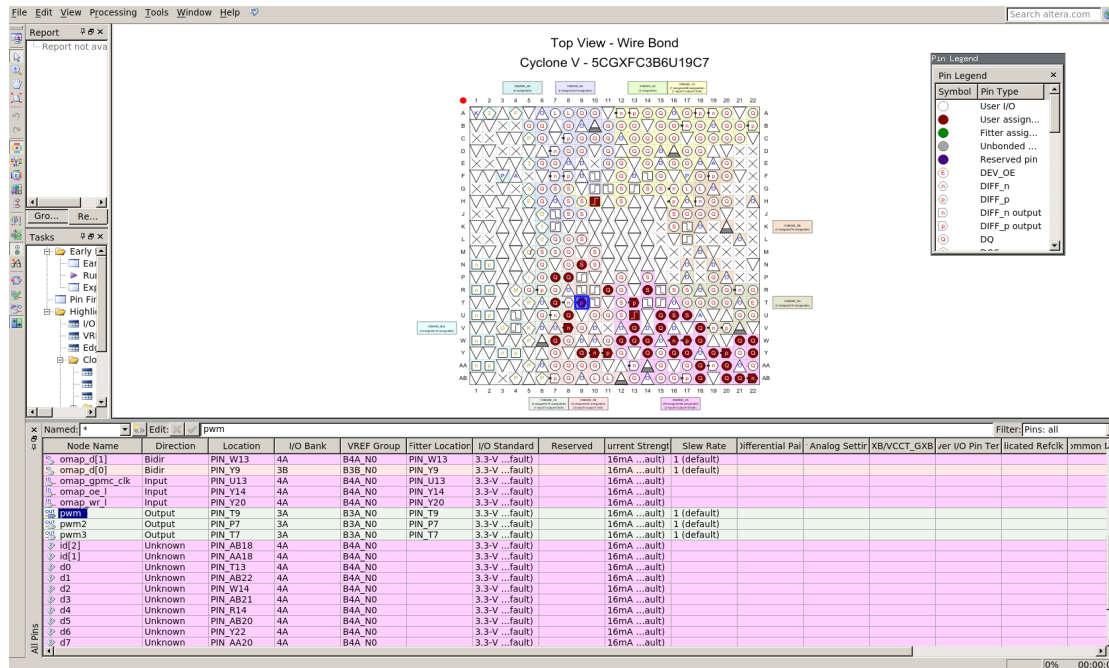


Figure 4.5 – Pin Planner in Quartus

After the Cyclone V has been configured, then we need to run the C server.

```
root@beaglebone:~/DEMO/PWM# ./BBB_pwm -debug
```

The C server will start by initialize everything it needs, then it will read the device ID from the FPGA. This is a code that we have assigned to the Cyclone V when we configured the pwm\_c5 project. If there are no errors ins our set up, then theC code will start sending the values of 50% PWM to each output , until it gets a new set of values from the TCP/IP connection.

**NOTE :** This interaction between the C server running on the BeagleBone processor and the Cyclone V, requires that the GPMC is working, so unsure that the device tree for the RedCape has been loaded.

The next step is to run the LabVIEW application in a Host Computer. The Host Computer has to be connected to the same network as the BeagleBone , or to have the required privileges to access the BeagleBone. The IP address of the BeagleBone can be found using the command ipconfig.

**NOTE :** The LabVIEW executable requieres the LabVIEW Run-Time Engine 2009 SP1 or compatible. If required, it can be downloaded from the National Instruments web page :

<http://www.ni.com/download/labview-run-time-engine-2009-sp1/1600/en/>

Once the LabVIEW program is execuced, enter the right IP and click on Connect. The button will chance color to green and will say Connected . Then you can move the values of each PWM signal by moving the dials. Values can also be changed

manually to modify the PWM settings.

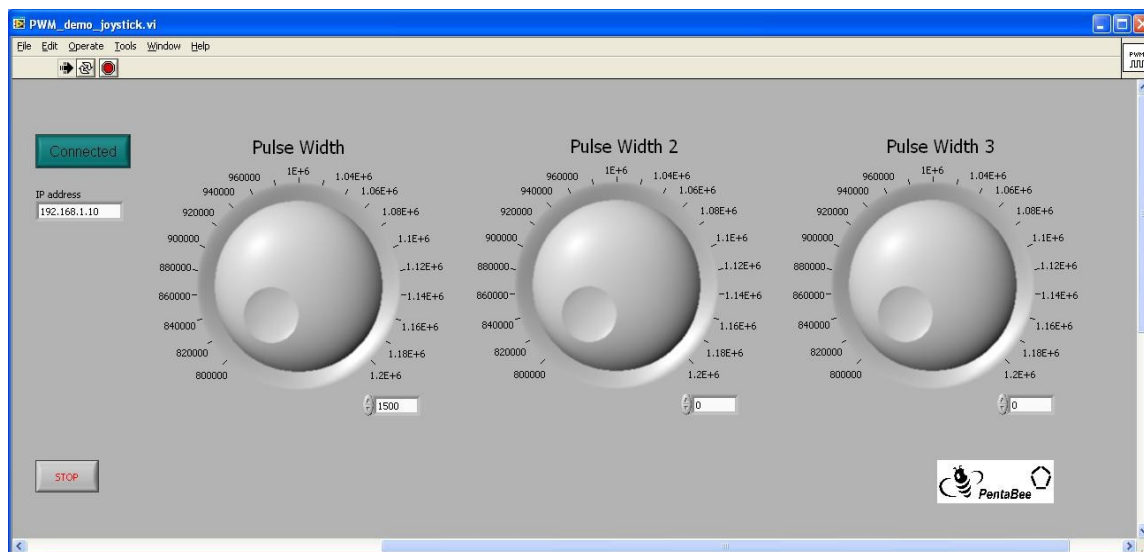


Figure 4.6 – LabVIEW GUI to control the PWM Signal



Figure 4.7 – Visualisation of PWM using user LEDs

## 5 – Appendix

### 5.1 - Connection Tables

**Table : FPGA-SATA Interconnections**

SATA_0	Cyclone V Ball	External
1		GND
2	Y4	
3	Y3	
4		GND
5	AA2	
6	AA1	
7		GND
8		CH_GND

SATA_1	Cyclone V Ball	External
1		GND
2	U2	
3	U1	
4		GND
5	W2	
6	W1	
7		GND
8		CH_GND

**Table : FPGA-Quad Clock Gen Interconnections**

Quad Clock Gen	Cyclone V Ball	External
CLK0A	V4	
CLK0B	U4	
CLK1A	L17	
CLK2A	R10	
CLK3A		Feedback clock
CLKIN	T12	

**Table : FPGA-PFC(Raspberry Pi Cam) Interconnections**

PFC – 15 Connector	Cyclone V Ball	External
1		GND
2	AA1	
3	AA2	
4		GND
5	W1	
6	W2	
7		GND
8	R1	
9	R2	
10		GND
11		
12		
13		I <sup>2</sup> C2 SCL
14		I <sup>2</sup> C2 SDA
15		VCC (3.3V)

**Table : FPGA- Pmods Interconnections**

Pmod -12 (P7)	Cyclone V Ball	External
1	T17	
2	T18	
3	R16	
4	R17	
5		GND
6		VCC (3.3V)
7	R20	
8	R19	
9	R21	
10	P19	
11		GND
12		VCC (3.3V)

Pmod -6 (P3)	Cyclone V Ball	External
1	E20	
2	F20	
3	H20	
4	H19	
5		GND
6		VCC (3.3V)



**Table 3 GPMC - FPGA I/O Interconnections**

FPGA		BeagleBone Black			
		BBB Headers		NAME	CPU
NAME	PIN	P8	P9		BALL
GPMC_D0	Y9	25	-	MMC1_DAT0	U7
GPMC_D1	W13	24	-	MMC1_DAT1	V7
GPMC_D2	W22	5	-	MMC1_DAT2	R8
GPMC_D3	W21	6	-	MMC1_DAT3	T8
GPMC_D4	W12	23	-	MMC1_DAT4	U8
GPMC_D5	U16	22	-	MMC1_DAT5	V8
GPMC_D6	V13	3	-	MMC1_DAT6	R9
GPMC_D7	U17	4	-	MMC1_DAT7	T9
GPMC_D8	V15	19	-	EHRPWM2A	U10
GPMC_D9	Y19	13	-	EHRPWM2B	T10
GPMC_D10	Y17	14	-	GPIO0_26	T11
GPMC_D11	W16	17	-	GPIO0_27	U12
GPMC_D12	W18	12	-	GPIO1_12	T12
GPMC_D13	V18	11	-	GPIO1_13	R12
GPMC_D14	Y16	16	-	GPIO1_14	V13
GPMC_D15	W17	15	-	GPIO1_15	U13
GPMC_A1	P8	45	-	LCD_DATA0	R1
GPMC_A2	V8	46	-	LCD_DATA1	R2
GPMC_A3	W7	43	-	LCD_DATA2	R3
GPMC_A4	Y10	44	-	LCD_DATA3	R4
GPMC_nCS0	Y11	26	-	GPIO1_29	V6
GPMC_nOE	Y14	8	-	TIMER7	T7
GPMC_NADV_ALE	U12	7	-	TIMER4	R7
GPMC_nWE	Y20	10	-	TIMER6	U6
GPMC_CLK	U13	18	-	GPIO2_1	V12
GPMC_BEOn	V19	9	-	TIMER5	T6
GPMC_WAIT0	J16	-	11	UART4_RXD	T17
GPMC_nCS1	K16	-	13	UART4_TXD	U17
FPGA_INT_N	L9	-	42	GPIO0_7	C18
FPGA_TDI	P5	-	27	GPIO3_19	C13
FPGA_TCK	V5	-	25	GPIO3_21	A14
FPGA_TDO	V3	-	23	GPIO1_17	V14
FPGA_TMS	R4	-	15	GPIO1_16	R13

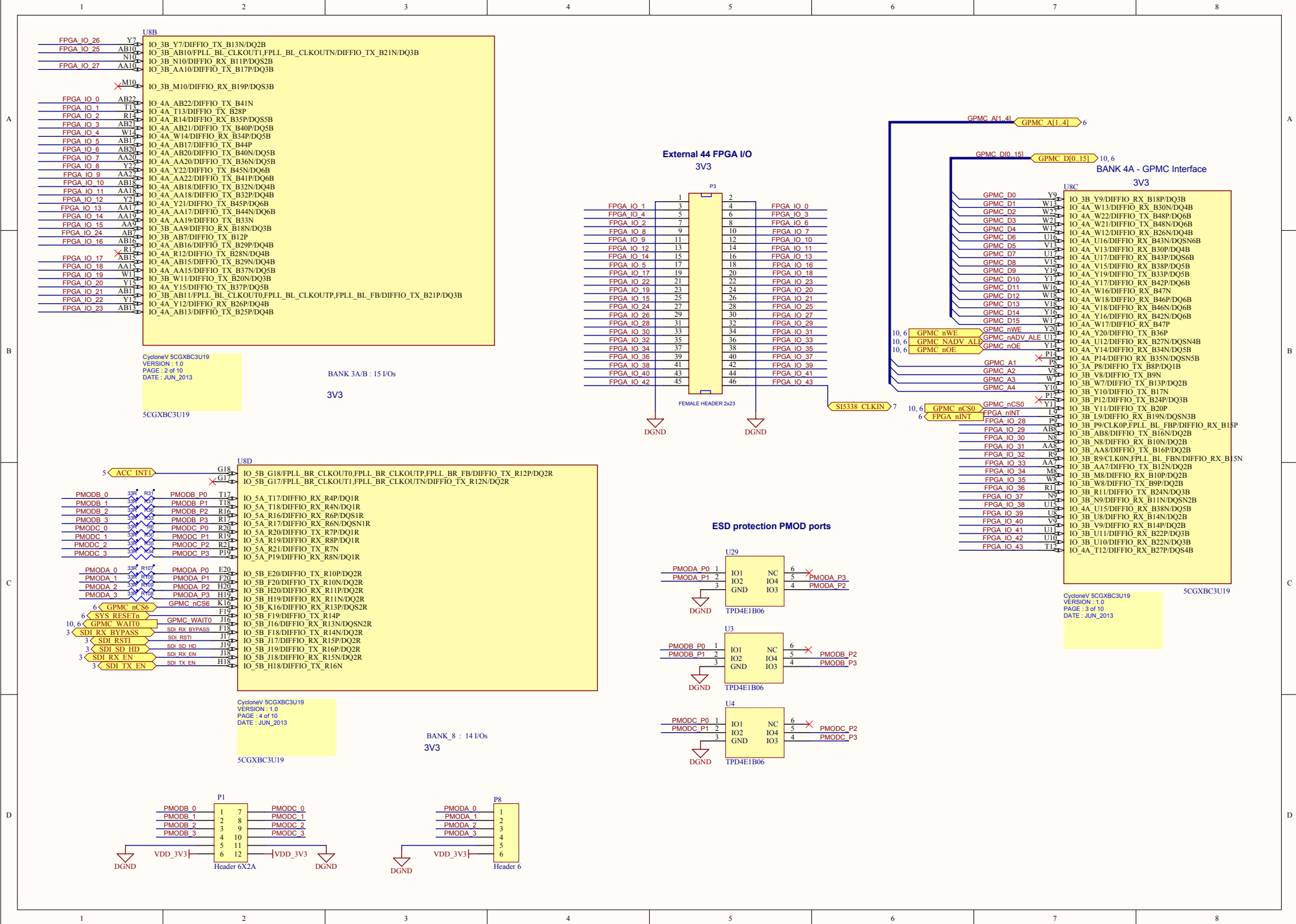
## Connectors of FPGA and P8 RedCape VA1

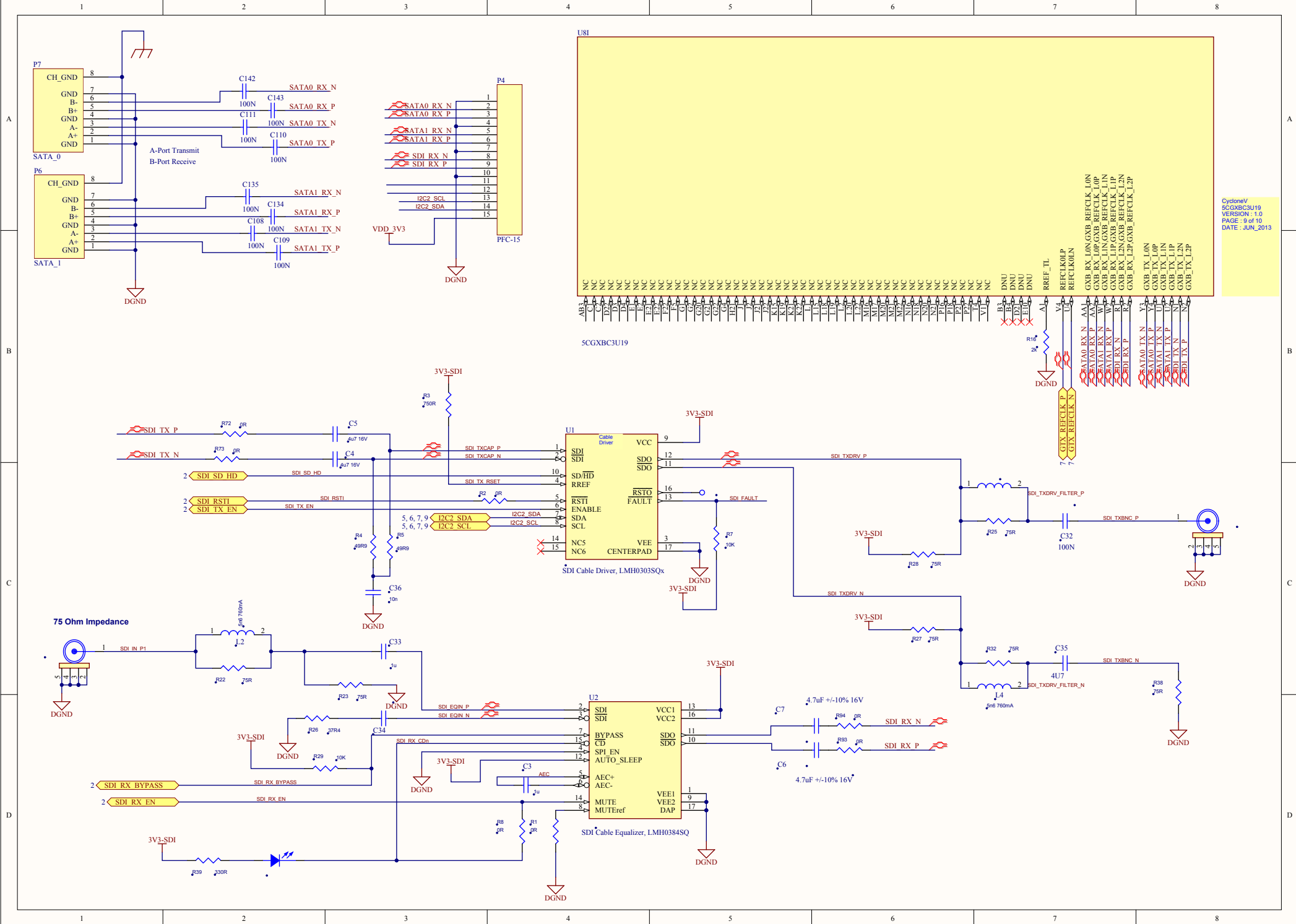
FPGA		P8	Usage
Name	Ball	PIN	
GND		1	
GND		2	
FPGA_IO_1	T13	3	
FPGA_IO_0	AB22	4	
FPGA_IO_4	W14	5	
FPGA_IO_3	AB21	6	
FPGA_IO_2	R14	7	
FPGA_IO_6	AB20	8	
FPGA_IO_8	Y22	9	
FPGA_IO_7	AA20	10	
FPGA_IO_9	AA22	11	
FPGA_IO_10	AB18	12	
FPGA_IO_12	Y21	13	
FPGA_IO_11	AA18	14	
FPGA_IO_14	AA19	15	
FPGA_IO_13	AA17	16	
FPGA_IO_5	AB17	17	
FPGA_IO_16	AB16	18	
FPGA_IO_17	AB15	19	
FPGA_IO_18	AA15	20	
FPGA_IO_22	Y12	21	
FPGA_IO_23	AB13	22	
FPGA_IO_19	W11	23	
FPGA_IO_20	Y15	24	
FPGA_IO_15	AA9	25	
FPGA_IO_21	AB11	26	
FPGA_IO_24	AB7	27	
FPGA_IO_25	AB10	28	
FPGA_IO_26	Y7	29	
FPGA_IO_27	AA10	30	
FPGA_IO_28	P9	31	
FPGA_IO_29	AB8	32	
FPGA_IO_30	N8	33	
FPGA_IO_31	AA8	34	
FPGA_IO_32	R9	35	
FPGA_IO_33	AA7	36	
FPGA_IO_34	M8	37	
FPGA_IO_35	W8	38	
FPGA_IO_36	R11	39	
FPGA_IO_37	N9	40	
FPGA_IO_38	U15	41	
FPGA_IO_39	U8	42	
FPGA_IO_40	V9	43	
FPGA_IO_41	U11	44	
FPGA_IO_42	U10	45	
FPGA_IO_43	T12	46	

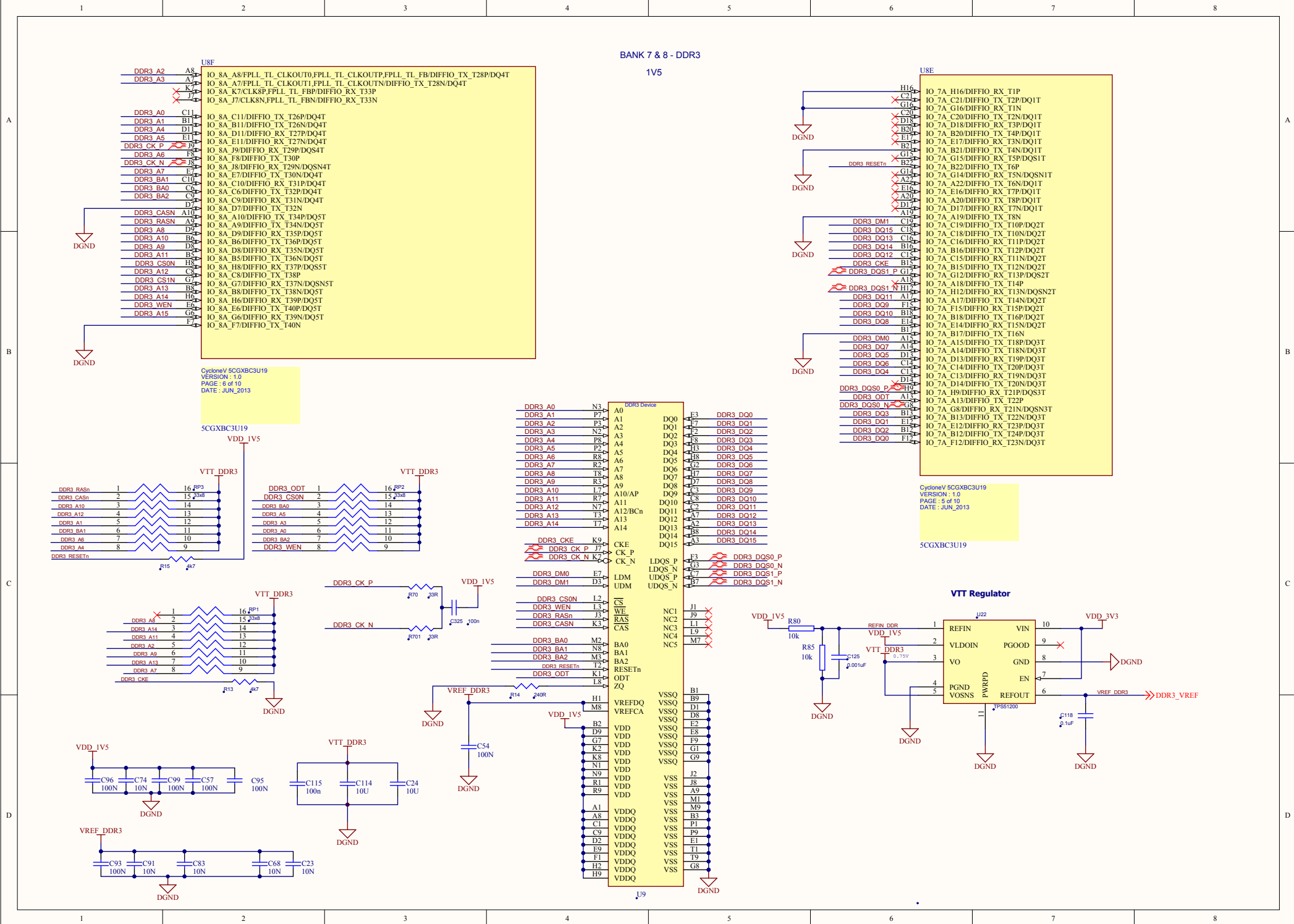
**Table : FPGA-DDR2 Interconnections**

FPGA		DDR2	
NAME	PIN	PIN	NAME
DDR2_A0	C11	M8	A0
DDR2_A1	B11	M3	A1
DDR2_A2	A8	M7	A2
DDR2_A3	A7	N2	A3
DDR2_A4	D11	N8	A4
DDR2_A5	E11	N3	A5
DDR2_A6	F8	N7	A6
DDR2_A7	E7	P2	A7
DDR2_A8	D9	P8	A8
DDR2_A9	D8	P3	A9
DDR2_A10	B6	M2	A10
DDR2_A11	B5	P7	A11
DDR2_A12	C8	R2	A12
DDR2_A13	B8	R8	RFU
DDR2_A14	H6	R3	RFU
DDR2_A15	G6	R7	RFU
DDR2_BA0	C6	L2	BA0
DDR2_BA1	C10	L3	BA1
DDR2_BA2	C9	L1	RFU
DDR2_DQ0	F12	G8	DQ0
DDR2_DQ1	E12	G2	DQ1
DDR2_DQ2	B12	H7	DQ2
DDR2_DQ3	B13	H3	DQ3
DDR2_DQ4	C13	H1	DQ4
DDR2_DQ5	D13	H9	DQ5
DDR2_DQ6	C14	F1	DQ6
DDR2_DQ7	A14	F9	DQ7
DDR2_DQ8	E14	C8	DQ8
DDR2_DQ9	F15	C2	DQ9
DDR2_DQ10	B18	D7	DQ10
DDR2_DQ11	A17	D3	DQ11
DDR2_DQ12	C15	D1	DQ12
DDR2_DQ13	C16	D9	DQ13
DDR2_DQ14	B16	B1	DQ14
DDR2_DQ15	C18	B9	DQ15
DDR2_CAS_n	A10	L7	CAS#
DDR2_CKE	B15	K2	CKE
DDR2_CS0_n	H8		
DDR2_CS1_n	G7	L8	CS#
DDR2_DM0	A15	F3	LDM
DDR2_DM1	C19	B3	UDM
DDR2_ODT	A13	K9	ODT
DDR2_RAS_n	A9	K7	RAS#
DDR2_WE_n	E6	K3	WE#
DDR2_CK_P	J9	J8	CK
DDR2_CK_N	J8	K8	CK#
DDR2_DQS0_P	H9	F7	LDQS
DDR2_DQS0_N	G8	E8	LDQS#/NU
DDR2_DQS1_P	G12	B7	UDQS
DDR2_DQS1_N	H12	A8	UDQS#/NU

## 5.2 Schematics



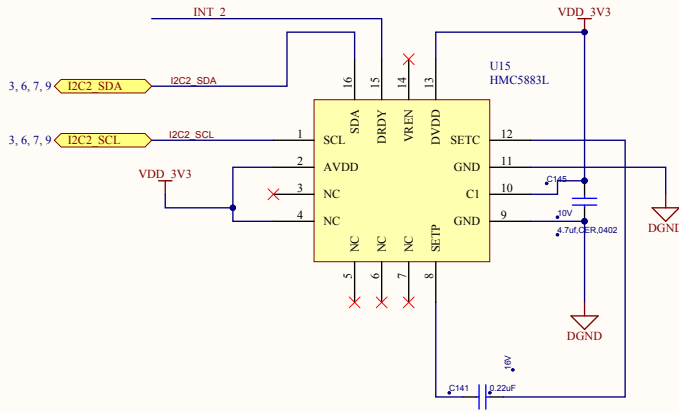






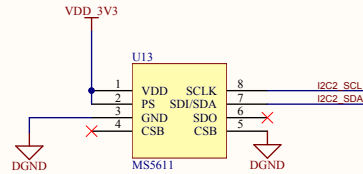
### 3D Magnetometer

I2C address: \$1E



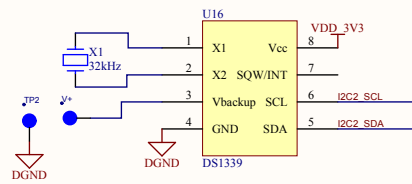
### Differential Pressure Sensor

I2C address: \$77



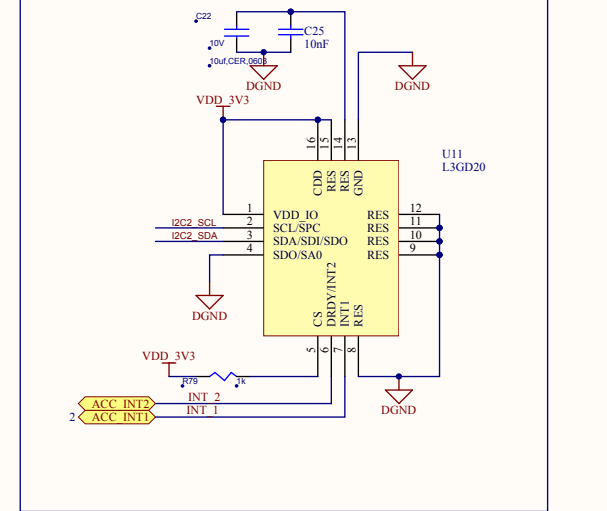
### Real Time Clock w/ Battery backup

I2C address: \$68



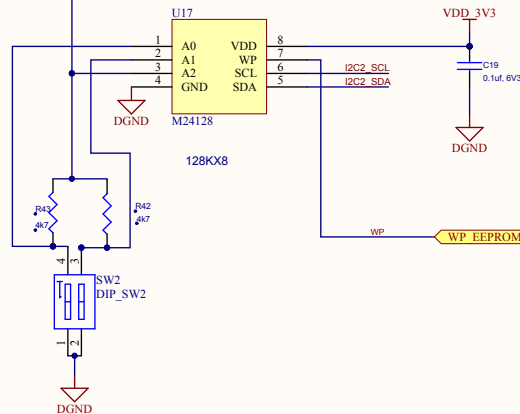
### Gyroscope

I2C address: \$6A



### Board ID EEPROM

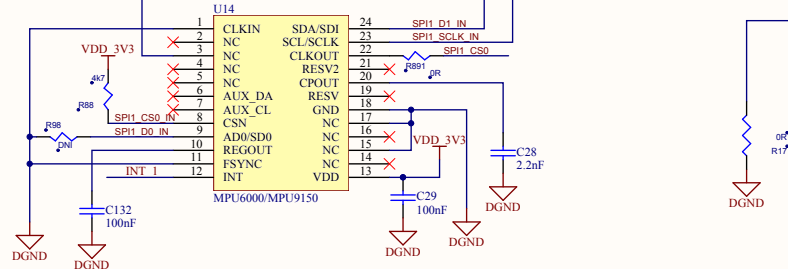
I2C address: \$54-57



### Integrated IMU

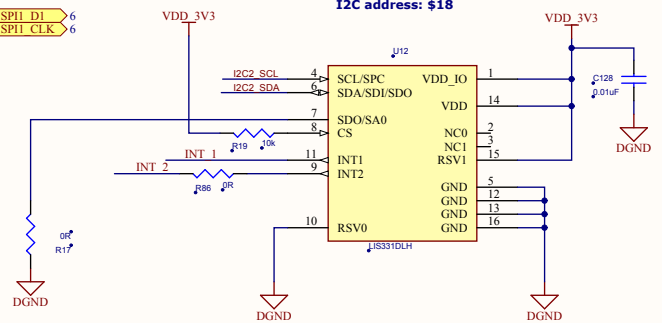
Uses SPI1 bus as per default

Optional I2C address: \$69/\$68

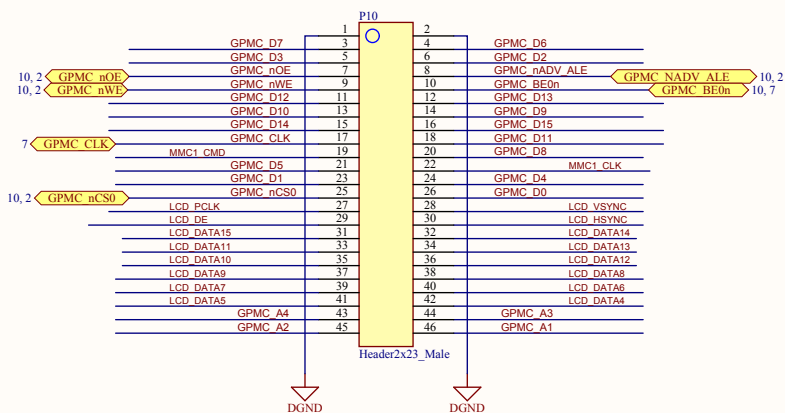


### Accelerometer

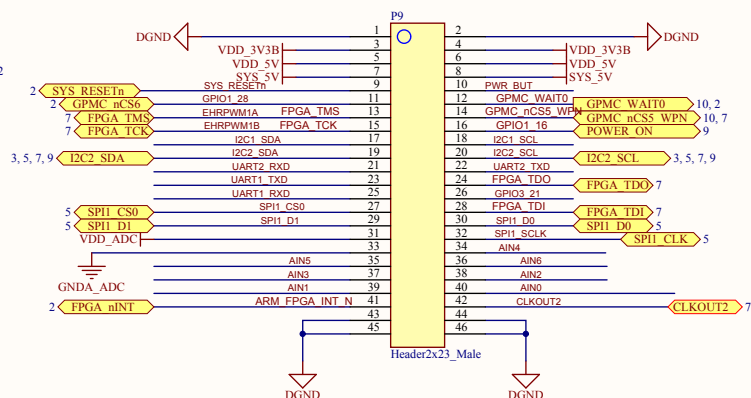
I2C address: \$18



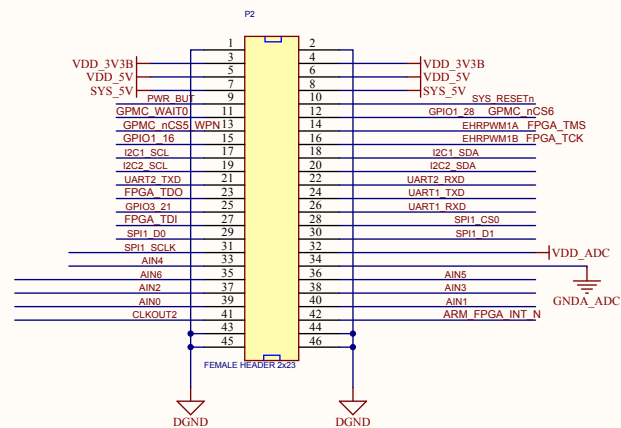
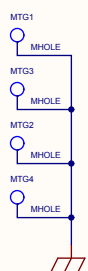
Resistor Network ADDR setting EEPROM without SW2 address is \$57



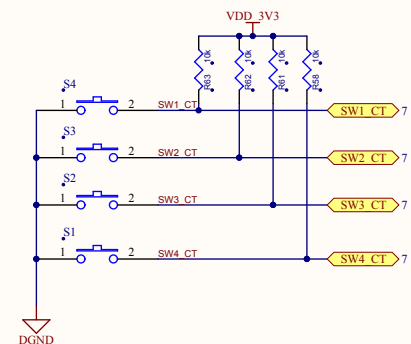
EXPANSION HEADER - Bottom Layer - Male



User Switches

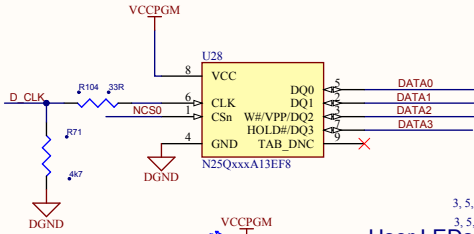


EXPANSION HEADER - Top Layer - Female

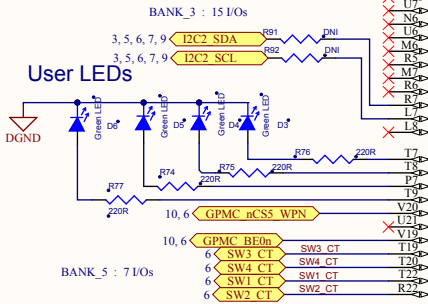


Configuration: AS active serial  
1-4bit EEPROM

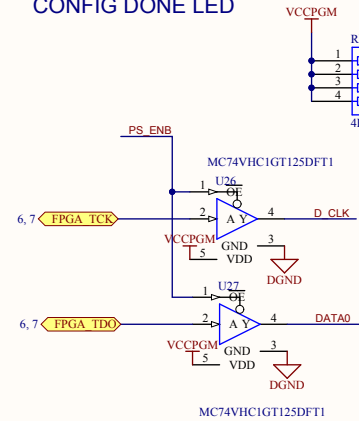
BANK 3A - Configuration/ GPIO  
3V3



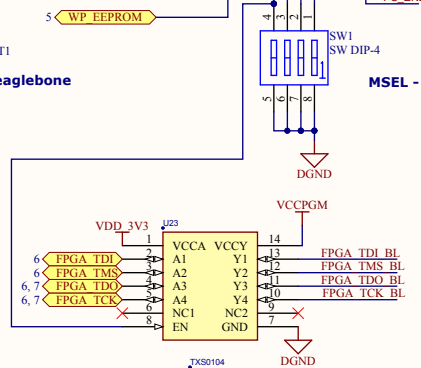
User LEDs



CONFIG DONE LED

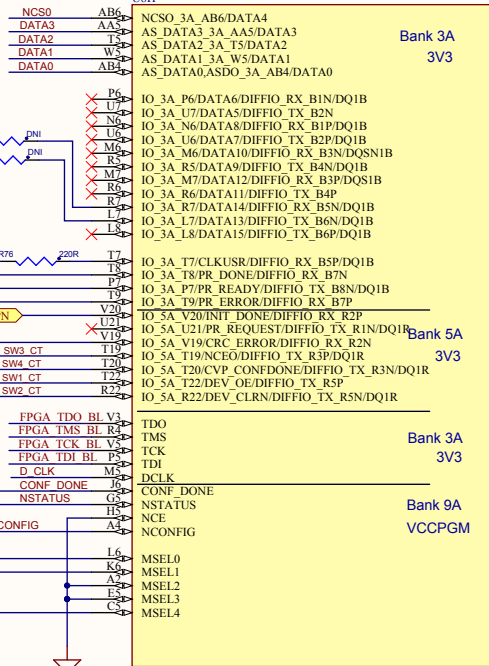


Configuration: Passive Serial using Beaglebone

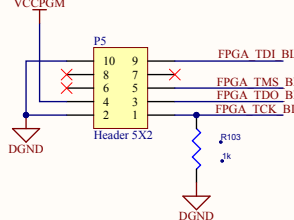


Configuration: JTAG using JRUNNER

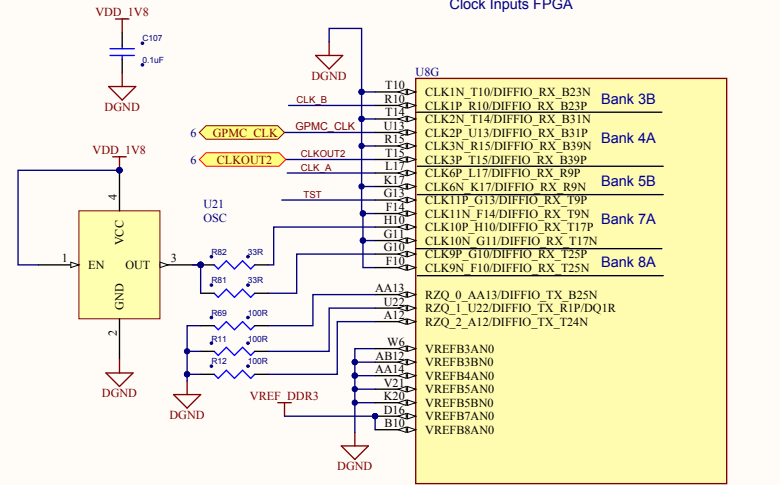
MSEL - JTAG and 4-bit parallel PS & AS



USB BLASTER

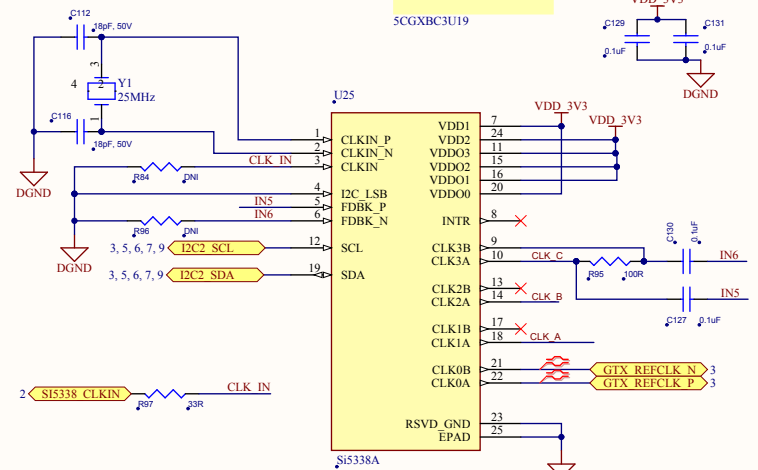


Clock Inputs FPGA



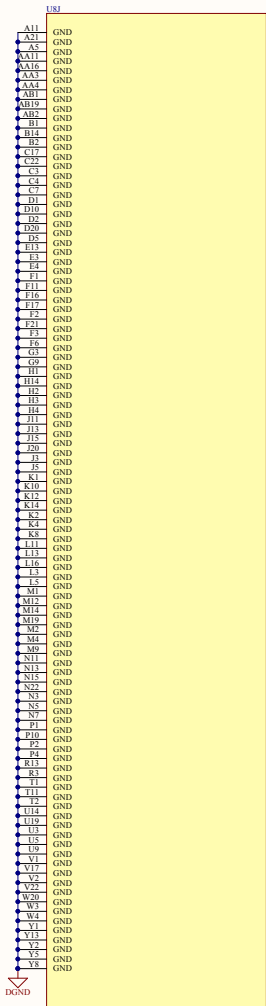
CycloneV SCGXBC3U19  
VERSION : 1.0  
PAGE : 7 of 10  
DATE : JUN\_2013

SCGXBC3U19



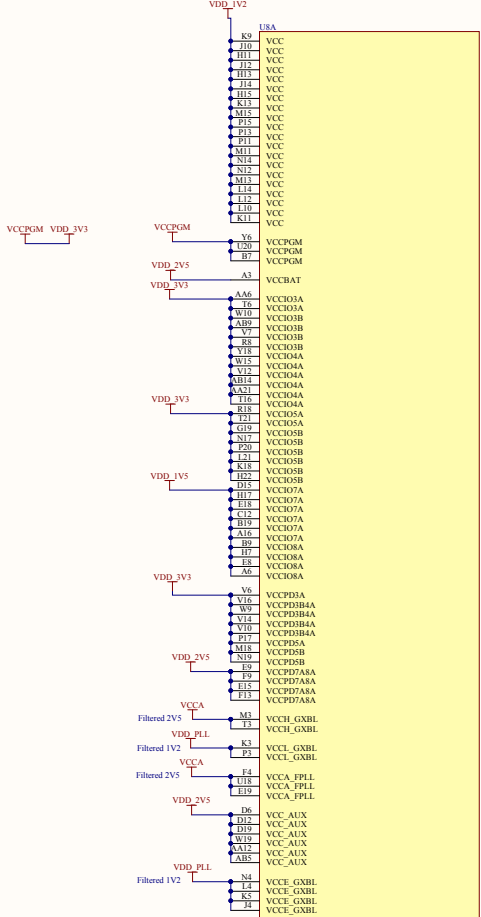
I2C address: \$70

FPGA CORE 1V2 Supply



Cytronix RCGXBCU19  
VERSION: 1.0  
PAGE: 10 of 10  
DATE: JUN\_2013

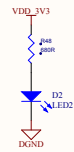
SCGXBCU19



Cytronix RCGXBCU19  
VERSION: 1.0  
PAGE: 1 of 10  
DATE: JUN\_2013

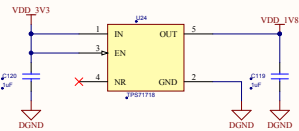
SCGXBCU19

### Power LED

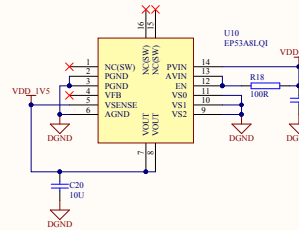


Populated for Stand Alone operation

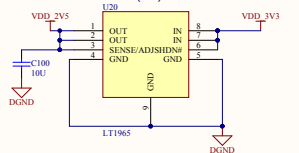
### 1V8 Supply



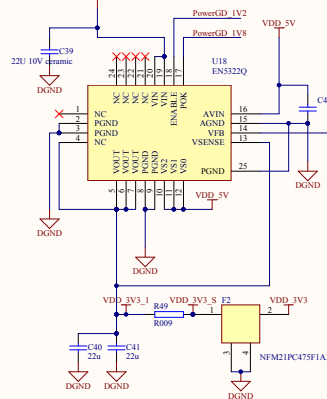
### 1V5 Supply DDR3 (1A)



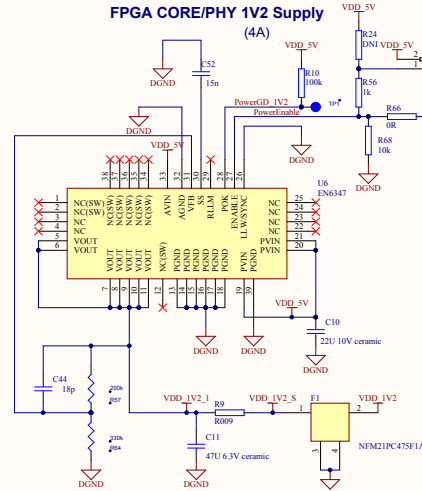
### 2V5 Supply FPGA (1A)



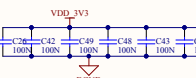
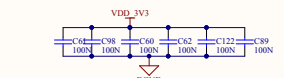
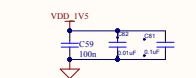
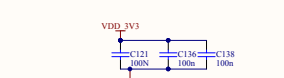
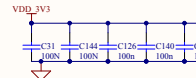
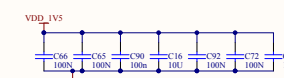
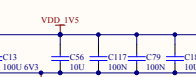
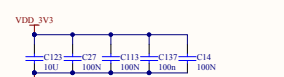
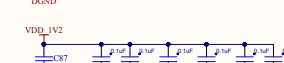
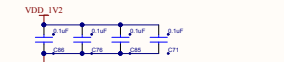
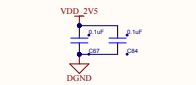
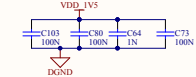
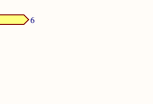
### 3V3 Supply (2A)



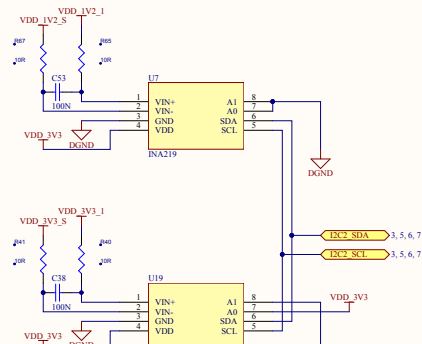
### FPGA CORE/PHY 1V2 Supply (4A)



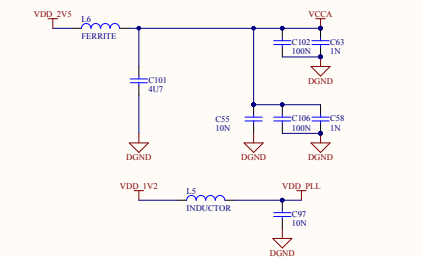
Insert Jumper for Stand Alone Operation



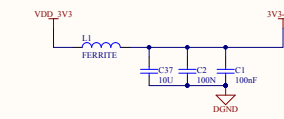
### Power/Voltage/Current Monitor



### FPGA\_PLL



### SDI\_supply



# NAND GPMC memory

