Final Report

SpiNNaker Group

Patrick Camilleri

9th February 2016

Contents

1 EMC testing documentation

Links http://learnemc.com/introduction-to-emc

1.1 EMC background

Electromagnetic compatibility (**EMC**) is the branch of electrical engineering concerned with the unintentional generation, propagation and reception of electromagnetic energy which may cause unwanted effects such as electromagnetic interference (EMI) or even physical damage in operational equipment. The goal of EMC is the correct operation of different equipment in a common electromagnetic environment.

EMC pursues two main classes of issue. **Emission** is the generation of electromagnetic energy, whether deliberate or accidental, by some source and its release into the environment. EMC studies the unwanted emissions and the countermeasures which may be taken in order to reduce unwanted emissions. The second class, **susceptibility** is the tendency of electrical equipment, referred to as the victim, to malfunction or break down in the presence of unwanted emissions, which are known as Radio frequency interference (RFI). **Immunity** is the opposite of susceptibility, being the ability of equipment to function correctly in the presence of RFI, with the discipline of "hardening" equipment being known equally as susceptibility or immunity. A third class studied is **coupling**, which is the mechanism by which emitted interference reaches the victim.

Interference mitigation and hence electromagnetic compatibility may be achieved by addressing any or all of these issues, i.e., quieting the sources of interference, inhibiting coupling paths and/or hardening the potential victims. In practice, many of the engineering techniques used, such as grounding and shielding, apply to all three issues.

1.2 Software description

2 Power measurement on the SpiNN4 board

2.1 Hardware used

2.1.1 ADC Arduino shield

2.1.2 Shunt amplifier circuits

2.2 Software used

2.2.1 Arduino Processing software

Processing is an open source language/development tool for writing programs in *other*computers. Useful when you want those other computers to "talk" with an Arduino, for instance to display or save some data collected by the Arduino.

Processing is an open source programming language and integrated development environment (IDE) built for the electronic arts, new media art, and visual design communities with the purpose of teaching the fundamentals of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks. The project was initiated in 2001 by Casey Reas and Benjamin Fry, both formerly of the Aesthetics and Computation Group at the MIT Media Lab. One of the stated aims of Processing is to act as a tool to get non-programmers started with programming, through the instant gratification of visual feedback. The language builds on the Java language, but uses a simplified syntax and graphics programming model. In 2012, they started the Processing Foundation along with Daniel Shiffman, who formally joined as a third project lead.

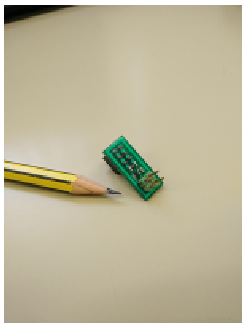


Figure 2.1: Shunt amplifier

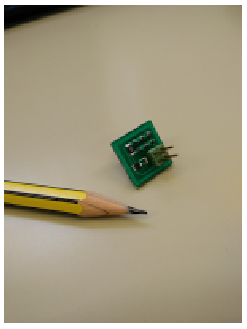


Figure 2.2: Shunt amplifier

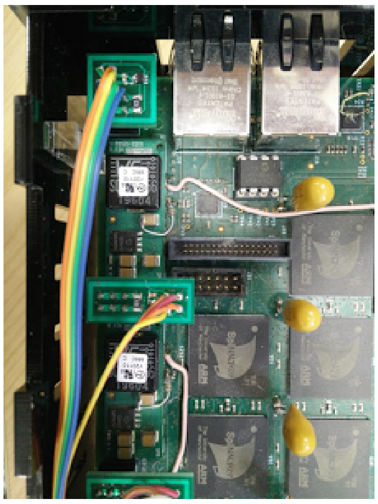


Figure 2.3: Shunt amplifiers mounted on SpiNN4 board

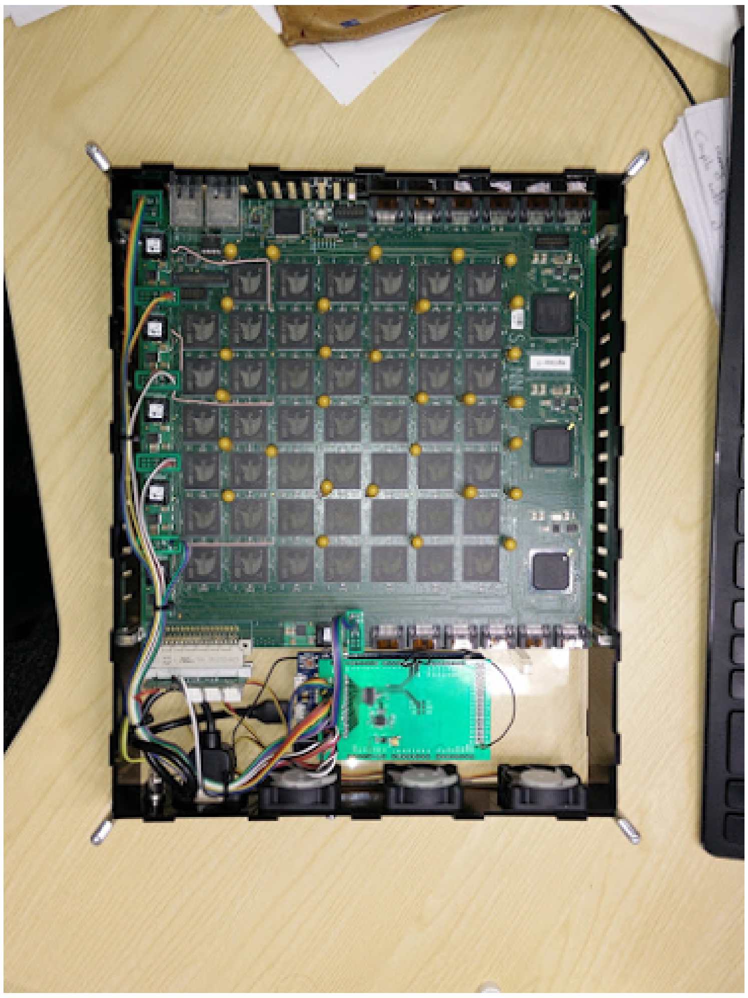


Figure 2.4: SpiNN4 board showing the shunt amplifiers and the ADC Arduino shield

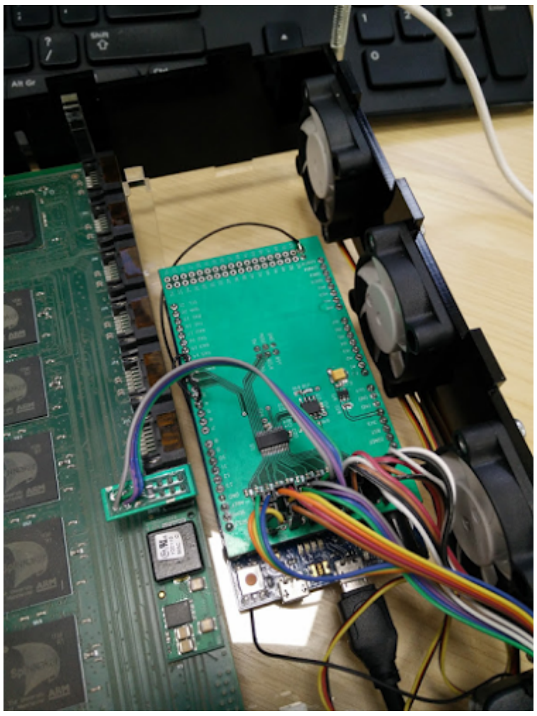


Figure 2.5: Close-up of the ADC Arduino shield

Features

Processing includes a *sketchbook*, a minimal alternative to an integrated development environment (IDE) for organizing projects.

Every Processing sketch is actually a subclass of the PApplet Java class which implements most of the Processing language's features.

When programming in Processing, all additional classes defined will be treated as inner classes when the code is translated into pure Java before compiling. This means that the use of static variables and methods in classes is prohibited unless you explicitly tell Processing that you want to code in pure Java mode.

Processing also allows for users to create their own classes within the PApplet sketch. This allows for complex data types that can include any number of arguments and avoids the limitations of solely using standard data types such as: int (integer), char (character), float (real number), and color (RGB, ARGB, hex).

3 Markov Chain Monte Carlo on SpiNNaker

https://darrenjw.wordpress.com/2010/04/28/mcmc-programming-in-r-python-java-and-c/

Markov chain Monte Carlo (MCMC) is a powerful simulation technique for exploring high-dimensional probability distributions. It is particularly useful for exploring posterior probability distributions that arise in Bayesian statistics. Although there are some generic tools (such as WinBugs and JAGS) for doing MCMC, for non-standard problems it is often desirable to code up MCMC algorithms from scratch. It is then natural to wonder what programming languages might be good for this purpose. There are hundreds of programming languages one could in principle use for MCMC programming, but it is necessary to use a language with a good scientific library, including good random number generation routines. I have used a large number of programming languages over the years, but these days I mostly program in R, Python,Java or C. I find each of these languages interesting and useful, with different strengths and weaknesses, and I find that no one of these languages dominates any of the others in all situations.

3.1 Example

For the purposes of this post we will focus on Gibbs sampling for a simple bivariate distribution defined on the half-plane x>0.

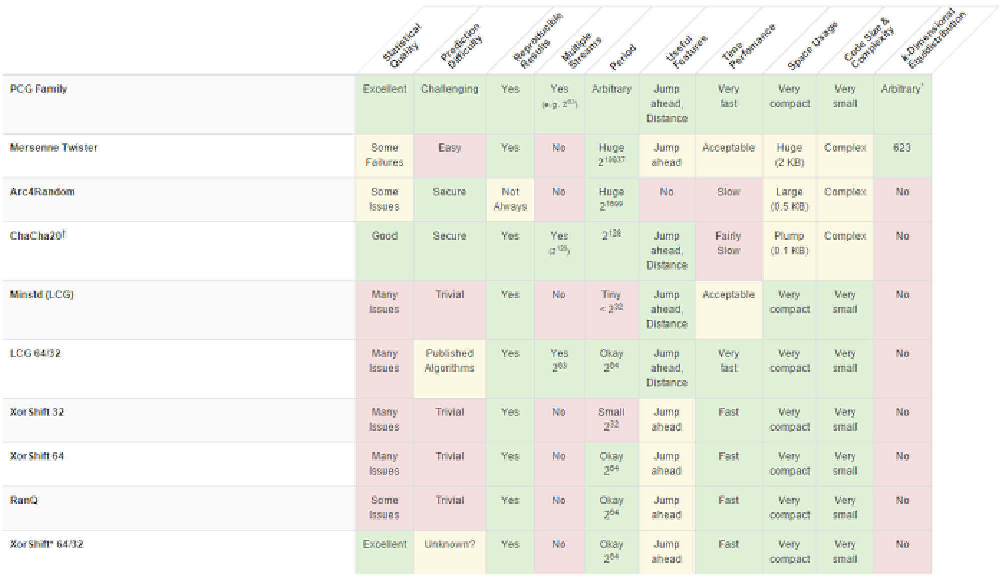
$$f(x, y) = kx^2 \exp\left(-xy^2 - y^2 + 2^y - 4x\right)$$

The statistical motivation is not important for this post, but this is the kind of distribution which arises naturally in Bayesian inference for the mean and variance of a normal random sample. Gibbs sampling is a simple MCMC scheme which samples in turn from the full-conditional distributions. In this case, the full-conditional for x is $\Gamma(3, y^2 + 4)$ and the full-conditional for y is N(1/(x+1), 1/(x+1)). The resulting Gibbs sampling algorithm is very simple and works very efficiently, but for illustrative purposes, we will consider generating 20,000 iterations with a "thin" of 500 iterations.

3.1.1 C

The language I have used most for the development of MCMC algorithms is C (usually strict ANSI C). C is fast and efficient, and gives the programmer lots of control over how computations are carried out. TheGSL is an excellent scientific library for C, which includes good random number generation facilities. A program for this problem is given below.

```
#include <math.h>
#include <stdlib.h>
#include <gsl/gsl_rng.h>
```



	Statistical Quality	Prediction Difficulty	Reproducible Results	Multiple Streams	Period	Useful Features	Time Perfomance	Space Usage	Code Size & Complexity	k-Dimensional Equidistribution
PCG Family	Excellent	Challenging	Yes	Yes (e.g. 2^{63})	Arbitrary	Jump ahead, Distance	Very fast	Very compact	Very small	Arbitrary'
Mersenne Twister	Some Failures	Easy	Yes	No	Huge 2^{19937}	Jump ahead	Acceptable	Huge (2 KB)	Complex	623
Arc4Random	Some Issues	Secure	Not Always	No	Huge 2^{1699}	No	Slow	Large (0.5 KB)	Complex	No
ChaCha20[†]	Good	Secure	Yes	Yes (2^{128})	2^{128}	Jump ahead, Distance	Fairly Slow	Plump (0.1 KB)	Complex	No
Minstd (LCG)	Many Issues	Trivial	Yes	No	Tiny $< 2^{32}$	Jump ahead, Distance	Acceptable	Very compact	Very small	No
LCG 64/32	Many Issues	Published Algorithms	Yes	Yes 2^{63}	Okay 2^{64}	Jump ahead, Distance	Very fast	Very compact	Very small	No
XorShift 32	Many Issues	Trivial	Yes	No	Small 2^{32}	Jump ahead	Fast	Very compact	Very small	No
XorShift 64	Many Issues	Trivial	Yes	No	Okay 2^{64}	Jump ahead	Fast	Very compact	Very small	No
RanQ	Some Issues	Trivial	Yes	No	Okay 2^{64}	Jump ahead	Fast	Very compact	Very small	No
XorShift* 64/32	Excellent	Unknown?	Yes	No	Okay 2^{64}	Jump ahead	Fast	Very compact	Very small	No

Figure 3.1: PCG family comparison

```c
#include <gsl/gsl_randist.h>

void main()
{
  int N=20000;
  int thin=500;
  int i,j;
  gsl_rng *r = gsl_rng_alloc(gsl_rng_mt19937);
  double x=0;
  double y=0;
  printf("Iter x y\n");
  for (i=0;i<N;i++) {
    for (j=0;j<thin;j++) {
      x=gsl_ran_gamma(r,3.0,1.0/(y*y+4));
      y=1.0/(x+1)+gsl_ran_gaussian(r,1.0/sqrt(x+1));
    }
    printf("%d %f %f\n",i,x,y);
  }
}
```

This can be compiled and run (on Linux) with commands like:

```
gcc -O2 -lgsl -lgslcblas gibbs.c -o gibbs
time ./gibbs > data.tab
```

PCG

http://www.pcg-random.org/

PCG is a family of simple fast space-efficient statistically good algorithms for random number generation. Unlike many general-purpose RNGs, they are also hard to predict.

- For the PCG family, arbitrary *k*-dimensional equidistribution (and the huge periods it implies) requires PCG's extended generation scheme.

† ChaCha entry based on an optimized C++ implementation of ChaCha, kindly provided by Orson Peters.

In statistics, **Markov chain Monte Carlo** (**MCMC**) methods are a class of algorithms for sampling from a probability distribution based on constructing a Markov chain that has the desired distribution as its equilibrium distribution. The state of the chain after a number of steps is then used as a sample of the desired distribution. The quality of the sample improves as a function of the number of steps.

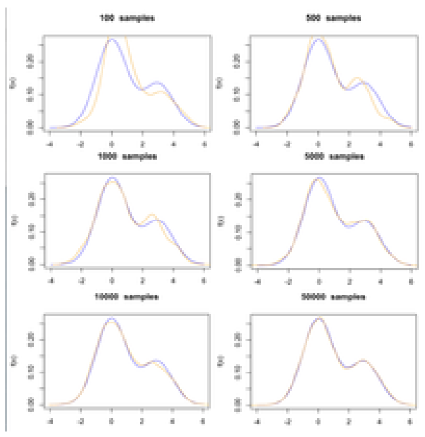


Figure 3.2: enter image description here

Convergence of the Metropolis-Hastings algorithm. MCMC attempts to approximate the blue distribution with the orange distribution

Random walk Monte Carlo methods make up a large subclass of MCMC methods.

3.2 Application domains

- MCMC methods are primarily used for calculating numerical approximations of multi-dimensional integrals, for example in Bayesian statistics, computational physics, computational biology and computational linguistics.[1][2]
- In Bayesian statistics, the recent development of MCMC methods has been a key step in making it possible to compute large hierarchical models that require integrations over hundreds or even thousands of unknown parameters.[3]
- They are also used for generating samples that gradually populate the rare failure region in rare event sampling.

3.3 Classification

3.3.1 Random walk Monte Carlo methods

Multi-dimensional integrals When an MCMC method is used for approximating a multi-dimensional integral, an ensemble of "walkers" move around randomly. At each point where a walker steps, the integrand value at that point is counted towards the integral. The walker then may make a number of tentative steps around the area, looking for a place with a reasonably high contribution to the integral to move into next.

Random walk Monte Carlo methods are a kind of random simulation or Monte Carlo method. However, whereas the random samples of the integrand used in a conventional

Monte Carlo integration are statistically independent, those used in MCMC methods are *correlate*d. A Markov chain is constructed in such a way as to have the integrand as its equilibrium distribution.

Examples Examples of random walk Monte Carlo methods include the following:

- Metropolis–Hastings algorithm: This method generates a random walk using a proposal density and a method for rejecting some of the proposed moves.
- Gibbs sampling: This method requires all the conditional distributions of the target distribution to be sampled exactly. When drawing from the full-conditional distributions is not straightforward other samplers-within-Gibbs are used (e.g, see [4][5][6]). Gibbs sampling is popular partly because it does not require any 'tuning'.
- Slice sampling: This method depends on the principle that one can sample from a distribution by sampling uniformly from the region under the plot of its density function. It alternates uniform sampling in the vertical direction with uniform sampling from the horizontal 'slice' defined by the current vertical position.
- Multiple-try Metropolis: This method is a variation of the Metropolis–Hastings algorithm that allows multiple trials at each point. By making it possible to take larger steps at each iteration, it helps address the curse of dimensionality.[7][8]
- Reversible-jump: This method is a variant of the Metropolis–Hastings algorithm that allows proposals that change the dimensionality of the space.[9] MCMC methods that change dimensionality have long been used in statistical physics applications, where for some problems a distribution that is a grand canonical ensemble is used (e.g., when the number of molecules in a box is variable). But the reversible-jump variant is useful when doing MCMC or Gibbs sampling over nonparametric Bayesian models such as those involving the Dirichlet process or Chinese restaurant process, where the number of mixing components/clusters/etc. is automatically inferred from the data.

4 Calibrating the SpiNNaker temperature sensors

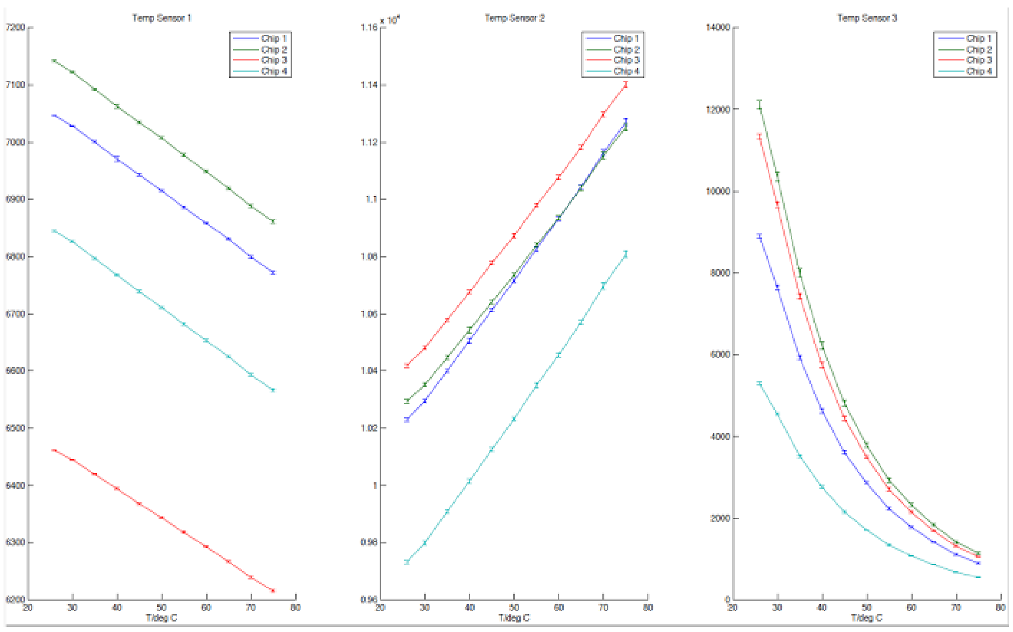


Figure 4.1: enter image description here

5 Event-based camera on cheap smartphone sensors and programmable hardware

5.1 Camera

5.2 OV7670

5.2.1 Introduction

OV7670 image sensor, small volume, low operating voltage, providing all functions of a single chip of VGA camera and image processor. Through SCCB bus control, the sensor can output the whole frame, sampling, and various resolution 8 bits of data. The product VGA image can reach up to a maximum of 30 frames per second. Users can completely control the image quality, data format and transmission mode. All the process of image processing functions can be through the SCCB programming interface, including gamma curve, white balance, saturation and chroma .

OmniVision image sensor has been in application of unique sensor technology, by reducing or eliminating the optical or electronic defect such as fixed pattern noise, tail, floating away, etc., to improve the quality of the image, and get the clear and stable color images.

Specifications:

- Photosensitive array: 640×480
- IO Voltage: 2.5 V to 3.0 V (internal LDO for nuclear power 1.8 V)
- Power operation: 60 mW/15 fps
- Sleep: 20 µA
- Temperature Operating: −30 °C to 70 °C
- Stable: 0 °C to 50 °C
- Output Formats (8): YUV/YCbCr4: 2:2 RGB565/555/444 GRB4: 2:2 Raw RGB Data

Figure 5.1: OV7670

Figure 5.2: OV5642

- Optical size: 1/6″
- FOV: 25 °C
- Maximum frame rate: 30 fps VGA
- Sensitivity: 1.3 V/(lux–sec)
- SNR: 46 dB
- Dynamic range: 52 dB
- View Mode: Progressive
- Electronic Exposure: 1 line to 510 line
- Pixel Size: 3.6 µm × 3.6 µm
- Dark current: 12 mV s^{-1} at 60 °C

Features:

- High sensitivity suitable for illumination applications
- Low voltage suitable for embedded applications
- Standard SCCB interface compatible with I2C interface
- RawRGB, RGB (GRB4:2:2, RGB565/555/444), YUV (4:2:2) and YCbCr (4:2:2) output format
- Supports VGA, CIF, and from a variety of sizes CIF to 40×30
- VarioPixel sub-sampling mode
- ISP has a compensation function to eliminate noise and dead pixels
- Support for image scaling
- Compensation for loss of optical lens
- 50/60 Hz automatic detection
- Saturation automatically adjust (UV adjustment)
- Automatically adjust edge enhancement
- Automatically adjust the noise reduction
- Automatically affect the control functions include: automatic exposure control, automatic gain control, automatic white balance, automatic elimination of light stripes, automatic black level calibration image quality control including color saturation, hue, gamma, sharpness

Read more: http://www.elecfreaks.com/store/ov7670-camera-module-p-705.html#ixzz3xzHrceGf

5.3 OV5642

5.3.1 Introduction

The OV5642 (color) image sensor is a low voltage, high-performance, 1/4-inch 5 megapixel CMOS image sensor that provides the full functionality of a single chip 5 megapixel (2592×1944) camera using OmniBSI™ technology in a small footprint package. It provides full-frame, sub-sampled, windowed or arbitrarily scaled 8-bit/10-bit images in various formats via the control of the Serial Camera Control Bus (SCCB) interface or MIPI interface. The OV5642 has an image array capable of operating at up to 15 frames per second (fps) in 5 megapixel resolution with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, defective pixel canceling, noise canceling, etc., are programmable through the SCCB interface, MIPI interface or embedded microcontroller. The OV5642 also includes a compression engine for increased processing power. In addition, Omnivision image sensors use proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical sources of image contamination, such as fixed pattern noise, smearing, etc., to produce a clean, fully stable, color image. The OV5642 has an embedded microcontroller, which can be combined with an internal autofocus engine and programmable general purpose I/O modules (GPIO) for external autofocus control. It also provides an anti-shake function with an internal anti-shake engine. For identification and storage purposes, the OV5642 also includes a one-time programmable (OTP) memory. Compared to its predecessor, the OV5642 has embedded TrueFocus™ Lite that enables extended depth of field (EDoF). The OV5642 supports both a digital video parallel port and a serial MIPI port. The MIPI and ISP interface can be used for a second camera sensor without requiring a dual serial port camera system.

Features:

- 1.4 µm OmniBSI technology
- ultra high performance
- embedded TrueFocus ISP enabling better denoise, sharpening, gamma correction and colour correction
- automatic image control functions:
  - automatic exposure control (AEC)
  - automatic white balance (AWB)
  - automatic band filter (ABF)
  - automatic 50/60 Hz luminance detection
  - automatic black level calibration (ABLC)
- programmable controls for frame rate, AEC/AGC 16-zone size/position/weight control, mirror and flip, scaling, cropping, windowing, and panning
- image quality controls: color saturation, hue, gamma, sharpness (edge enhancement), lens correction, defective pixel canceling, and noise canceling
- support for output formats: RAW RGB, RGB565/555/444, CCIR656, YUV422/420, YCbCr422, and compression
- support for auto focus control (AFC)
- standard serial SCCB interface
- MIPI serieal input and output interface
- programmable I/O drive capability
- support for mechanical shutter, ND filter and IRIS control
- built-in 1.5 V regulator for core

Specifications:

- active array size: 2592×1944
- power supply:
  - core: 1.5 V ±5% (internal regulator)
  - analog: 2.6 V – 3.0 V
  - I/O: 1.71 V – 3.0 V
- power requirements:
  - active: 270 mA

  - standby: 25 µA
- temperature range:
  - operating: −30 °C to 70 °C
  - stable image: 0 °C to 50 °C
- lens sie: 1/4″
- lens chief ray angle: 24 deg non-linear
- input clock frequency: 6-54 MHz
- shutter: rolling shutter
- maximum image transfer rate:
  - 5 megapixel (2592×1944): 15 fps (and any size scaling down from 5 megapixel)
  - 1080p (1920×1080): 30 fps
  - 720p (1280×720): 60 fps
  - VGA (640×480): 60 fps
  - QVGA (320×240): 120 fps
- sensitivity: 680 mV/(lux−s)
- S/N ratio: 36 dB
- dynamic range: 68 dB
- pixel size: 1.4 µm × 1.4 µm
- image area: 3673.6 µm × 2738.4 µm
- package dimensions: 6945 µm × 6696 µm
- die dimensions: 6960 µm × 6710 µm

5.4 Zybo/Zedboard implementation

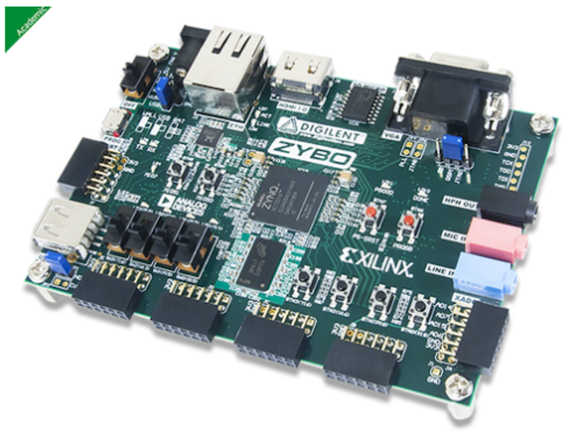


Figure 5.3: Zybo board

5.5 Vivado design

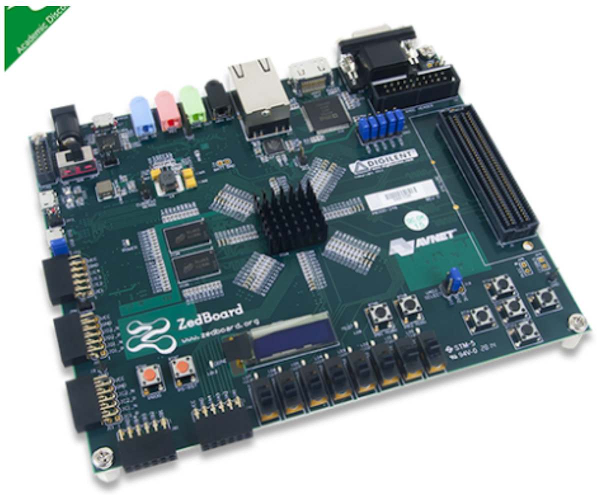
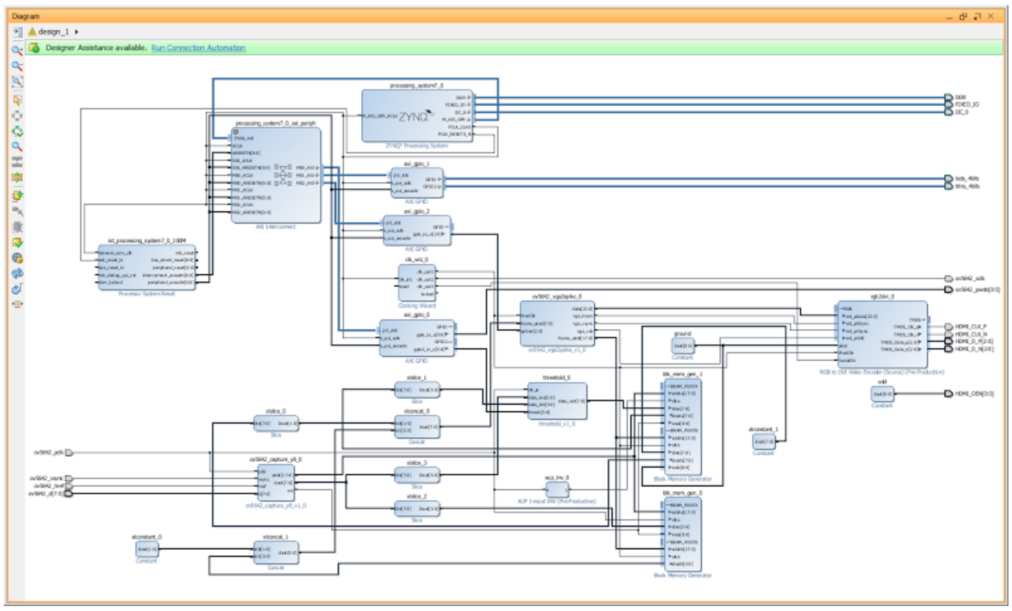


Figure 5.4: Zedboard



Designer Assistance available. Run Connection Automation

Figure 5.5: Vivado design