UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATION ENGINEERING

Caner Cuhac

CAMERA INTEGRATION TO WIRELESS SENSOR NODE

Master´s thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 24 May, 2011.

Supervisor	Mohammed Elmusrati
Instructor	Reino Virrankoski

TABLE OF CONTENTS

ABBREVATIONS

API	Application Programming Interface
3D	Three Dimensional
bmp	Bitmap Image File
DC	Direct Current
DRAM	Dynamic Random Access Memory
EMC	External Memory Controller
FIFO	First In First Out
fps	Frames per Second
GND	Ground
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuit
IDC	Insulation-displacement connector
JTAG	Joint Test Action Group
MSB	Most Significant Bit
PC	Personal Computer
PCB	Printed Circuit Board
PWM	Pulse Width Modulated
RGB	Red Green Blue
RS232	Recommended Standard 232
SCCB	Serial Camera Control Bus
SPI	Serial Peripheral Interface Bus
SRAM	Static Random Access Memory
USB	Universal Serial Bus
VDD	Positive Supply Voltage
WSAN	Wireless Sensor Actuator Network
WSN	Wireless Sensor Network

UNIVERSITY OF VAASA	
Faculty of technology	
Author:	Caner Çuhac
Topic of the Thesis:	Camera Integration to Wireless Sensor Node
Supervisor:	Mohammed Elmusrati
Instructor:	Reino Virrankoski
Degree:	Master of Science in Technology
Department:	Department of Computer Science
Degree Programme:	Degree Programme in Information Technology
Major of Subject:	Telecommunications Engineering
Year of Entering the University:	2009
Year of Completing the Master's Thesis:	2011 **Pages:** 78

ABSTRACT

A wireless sensor node with a vision sensing and image processing capabilities has a great utilisation potential in many industrial, healthcare and military applications. University of Vaasa has recently been developing a wireless sensor node called UWASA Node. It is a generic, modular and stackable wireless sensor platform. This work aims to integrate a camera module to UWASA Node and focuses on hardware design, software development, and easy image processing methods. Since the design is intended to prove the feasibility of the image processing in UWASA Node, a test board has been developed and integrated to a development kit which reflects the same behaviour as the sensor node platform. The new hardware and software has been designed and tested to verify vision sensor adaptation, image processing, and feature extraction in wireless sensor nodes. Due to the resource limited nature of the wireless sensor nodes, some new methods are introduced to achieve fast and efficient image processing. In summary, the hardware structure of the camera module and its working principles are designed explained, data handling and image processing methods are discussed, finally the achieved results are presented.

KEYWORDS: Camera, Image Processing, Wireless Sensor

1. INTRODUCTION

Wireless sensor nodes have a very wide application range in industrial automation, healthcare, and military applications thus it is a very promising technology with a great utilisation potential. A wireless sensor node with vision acquisition and processing capability introduces new wide application range for wireless networks. In automation systems decision making based on visual information greatly enhances the functionality because machine vision opens a wide door for information gathering about the environment of the network.

University of Vaasa has recently been developing a wireless sensor node called UWASA Node. It is a generic, modular and stackable wireless sensor platform (Yiğitler, Virrankoski & Elmusrati: 1). This work introduces a slave camera module integration to UWASA Node. Camera module is basically composed of a camera board and memory devices. The adapted camera board belongs to an already existing embedded vision system CMUcam3.

In order to adapt the camera board and the sensor node, a new hardware interface has been designed. The new interface basically contains a voltage converter, some logic components and memory devices. A depiction of the proposed hardware architecture is given in Figure 1.

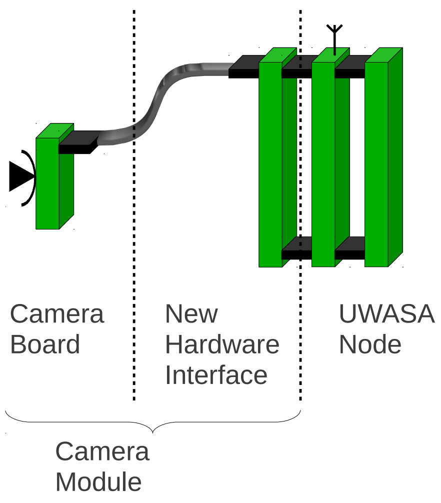


Figure 1. Hardware architecture of the vision sensor equipped UWASA Node.

In order to prove the feasibility of this system, the new hardware interface has been produced as a prototype PCB, that is, test board. Similarly, a development kit which reflects all the properties and peripherals of the UWASA Node has been used for prototyping purposes. The new designed test board gets the image data from the vision sensor, properly stores it, then the processor handles the rest.

Apart from designing a system for image storage and acquisition, image processing capabilities are also tested in this work. After capturing the original image, monochrome transformation, convolution, image gradient calculation, and edge detection algorithms have been verified.

Due to the limited computation power of the wireless sensor nodes, some new methods based on existing methods are introduced. For example in a part of the algorithm, an image processing operator somehow has to calculate a square root of a big number, instead of calculating the precise number and complex mathematical operations, an iterative method has been chosen for easy and fast computation.
Finally, the feasibility of image processing inside the wireless sensor node has been verified and the results are presented.

2. THEORY AND BACKGROUND INFORMATION

This section provides necessary background information related to the concepts and solutions used in the work.

2.1. Bayer filter

A photosensor is a semiconductor device that regulates its output regarding to the light intensity that falls onto its surface. A standard photosensor can not detect a specific colour sharply because the ions on its surface gets excited by a wider light range.

Bayer filter distributes the primary colours of the RGB colour space onto different adjacent photosensors. This allows reception of those primary colours in a form of two dimensional array. Elements of this two dimensional array in Bayer filter are oriented regarding to Figure 2.

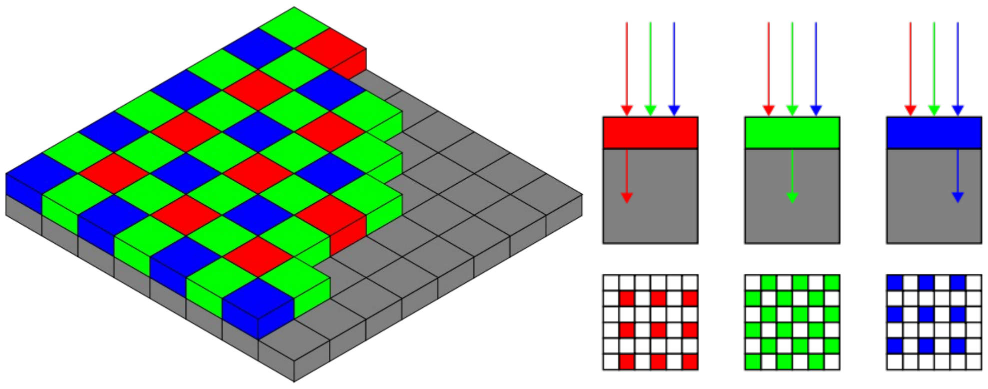


Figure 2. Bayer filter (Wikipedia).

In front of each photosensor lies a colour filter that lets only specific wavelength to fall onto the desired array element. Collection of the information from each semiconductor photosensor in a matrix forms the RGB image.

As can be noticed, the number of green pixels are twice of those red or blue pixels. The reason for that is the human eye is more sensitive to green colour than red or blue colours. Using those primary colours it is possible to represent any others.

2.2. Single server exponential queuing system

For a single server queuing system in which the arrivals are Poisson distributed random process with an intensity parameter λ, let λ be the unit arrival rate. In that case the mean time between two arrivals is 1/λ. Each arriving unit enters the service if the queue is empty, and it has to join to the end of the queue if there is any unit in the system. When the service is complete, the unit that has been serviced leaves the system and the next unit in the queue enters the service. If the service rate here is represented by μ, then the mean service time is 1/μ (Sheldon 2010: 502–504).

Assuming an infinite queue, let P_n, for n = 0, 1, 2, ... be the probability that an arriving unit finds n units in the queue. Here the queue is defined to be in state n if there n units waiting. If a new unit arrives the state of the queue will jump from n to n+1. It is clear that the rate at which the queue enters state n is equal to the rate that it leaves state n. Hence the equality for the start of the queue can be written as:

$$\lambda P_0 = \mu P_1 \tag{1}$$

This equation means that state 0 can increase by an arrival, and state 1 can decrease by a service. The queue can leave the state either by an arrival or by a service completed. Thus the rate at which the queue changes its state is $\lambda+\mu$. Here the proportion of the time that the process is in state 1 is equal to P_1 so the rate at which the queue leaves state 1 is equal to $P_1(\lambda+\mu)$. On the other hand, a state can either be entered by an arrival or by a departure, that is, state 1 can be entered from state 0 or from state 2. Formally this rate can be expressed as $\lambda P_0 + \mu P_2$. Since the rate the queue leaves state 1 is equal to the rate that it is entered from 0th or 2nd state:

$$P_1(\lambda+\mu) = \lambda P_0 + \mu P_2 \tag{2}$$

Then the equations for this queuing system are:

$$\begin{aligned} \lambda P_0 &= \mu P_1, && if \quad n = 0 \\ (\lambda + \mu) P_n &= \lambda P_{n-1} + \mu P_{n+1}, && if \quad n \geq 1 \end{aligned} \tag{3}$$

Those equations are called balance equations. Equation 3 can be expressed as:

$$\begin{aligned} P_1 &= \frac{\lambda}{\mu} P_0, && n = 0 \\ P_{n+1} &= \frac{\lambda}{\mu} P_n + \left(P_n - \frac{\lambda}{\mu} \; P_{n-1} \right), && n \geq 1 \end{aligned} \tag{4}$$

Solving those equations in terms of P_0:

$$P_0 = P_0 \tag{5}$$

$$P_1 = \frac{\lambda}{\mu} P_0 \tag{6}$$

$$P_2 = \frac{\lambda}{\mu} P_1 + \left(P_1 - \frac{\lambda}{\mu} P_0 \right) = \frac{\lambda}{\mu} P_1 = \left(\frac{\lambda}{\mu} \right)^2 P_0 \tag{7}$$

$$P_3 = \frac{\lambda}{\mu} P_2 + \left(P_2 - \frac{\lambda}{\mu} P_1 \right) = \frac{\lambda}{\mu} P_2 = \left(\frac{\lambda}{\mu} \right)^3 P_0 \tag{8}$$

$$P_{n+1} = \frac{\lambda}{\mu} P_n + \left(P_n - \frac{\lambda}{\mu} P_{n-1} \right) = \frac{\lambda}{\mu} P_n = \left(\frac{\lambda}{\mu} \right)^{n+1} P_0 \tag{9}$$

Using the fact that the sum of those probabilities equals to 1:

$$\sum_{n=0}^{\infty} P_n = 1 = \sum_{n=0}^{\infty} \left(\frac{\lambda}{\mu} \right)^n P_0 = \frac{P_0}{1 - \lambda / \mu} \tag{10}$$

Hence,

$$\begin{aligned} P_0 &= 1-\frac{\lambda}{\mu}, & n = 0 \\ P_n &= \left(\frac{\lambda}{\mu}\right)^n\left(1-\frac{\lambda}{\mu}\right), & n \geq 1 \end{aligned} \tag{11}$$

As it is stated before, λ is the unit arrival rate. For this hardware design, λ can be related with data arrival rate from the camera. In other words, the data output rate of the camera. Similarly the service rate μ can be related with the data handling rate of the processor. In other words, data input rate of the processor.

These equations are very important for embedded systems because they prove that the destination must handle the data faster than the source can output. If the processor would not have the capacity to handle the data faster than the camera outputs, λ would be greater than μ and that would lead the above equation to be a negative probability which is impossible. Theoretically this situation indicates that the queue reaches to infinity. In practice that would cause data loss.

Lastly, in a system where μ is greater than λ, the queue length is given by:

$$L = \sum_{n=0}^{\infty} n P_n \tag{12}$$

2.3. Two dimensional discrete convolution

Convolution in image processing is a two dimensional discrete operation applied usually for image transformations. The operands of the convolution are the convolution kernel and the concerned image. Convolution kernel is an array that determines what the convolution operation specifically does over the image. In most cases it is a fixed sized two dimensional array with an anchor point which is typically located in the centre of the array. A three by three convolution kernel is depicted in Figure 3 below.

-1	2	-1
2	5	2
-1	2	-1

Figure 3. A convolution kernel. The anchor point is usually located in the centre of the array.

The result of the convolution for a certain pixel location is computed by first placing the anchor point of the kernel on a pixel while the rest of the values around correspond to the overlapping pixels. Each element of the kernel is then multiplied with its matching pixel and the results are added together. This gives the convolution value at that point. Convolution operation is repeated for every pixel by sweeping the kernel over the entire image (Bradski & Kaehler 2008: 145).

If the image is represented by A(x, y) and the kernel by K(i, j). Assuming that the anchor point is located at (a, b) of the kernel coordinates, the convolution C(x, y) can be written as:

$$C(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} A(x+i-a,\ y+j-b)\, K(i,j) \tag{13}$$

Where M and N represents the horizontal and vertical sizes of the kernel respectively. Since for each pixel, the number of multiplications is equal to the kernel size, chosen kernel dimensions directly effects the required computational power. In advanced image processing library APIs, these operations are optimised.

2.4. Gradient and the Sobel operator

Sobel operator is a discrete differential operator that computes an approximation to the first derivative of the image at given location. It uses two dimensional convolution and some mathematical operations to compute the result.

The output of the Sobel operator is the vectorial sum of the horizontal and vertical gradient vectors at the given point. The values of the horizontal and vertical gradient vectors here are the calculated respective to the image intensity, that is, monochrome image.

According to (Kanopoulos, Vasanthavada & Baker: 358–359) Sobel operator uses two convolution kernels, one for horizontal derivation and another for vertical derivation. Both arrays are convolved with the original image to calculate the approximation to the derivations. If the image part that lays under the convolution kernels is represented with the matrix A, the horizontal gradient value G_x and the vertical gradient value G_y at that point are:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad and \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \tag{14}$$

where * denotes the convolution operation. Since the calculated gradient values are the lengths of two orthogonal vectors, the magnitude of the gradient can be calculated by:

$$G = \sqrt{G_x^2 + G_y^2} \tag{15}$$

and the angle of the gradient vector is:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{16}$$

The disadvantage of the Sobel operator is, it calculates rather inaccurate image gradient approximations. The reason of this handicap is the usage of integer values and the kernel size which is limited to 3x3. On the other hand, for many applications it provides satisfactory results in practice.

2.5. Babylonian square root method

Assuming that there is a positive number S that the square root of S is unknown, according to (Fowler & Robson: 367–369) Babylonian Square Root method approximates to the square root of S by sequential iterations with simple operations. In order to start iteration, an initial starting value x_0 should be chosen and placed in the equation. Here x_0 is a positive real valued initial number that will approach to the square root with each iteration. The better estimation approximates to the result faster and more accurate. Using this technique square root of S is calculated by:

$$x_0 \approx \sqrt{S} \tag{17}$$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right) \tag{18}$$

$$\sqrt{S} = \lim_{n \to \infty} x_n \tag{19}$$

where n is the number of iterations.

3. WIRELESS SENSOR NETWORKS AND IMAGE PROCESSING

This chapter covers wireless sensor networks, importance of vision sensors for those networks, hardware structure of the UWASA Node, image processing and feature extraction concepts.

3.1. Wireless sensor networks with vision sensors

Wireless sensor networks are formed of multiple wireless sensor nodes communicating with each other. WSNs offer ideal solutions to many application needs in industrial, military, and healthcare areas. These applications are mainly based on sensation, actuation, communication and network integration.

WSANs are formed of low cost, low power, short distance and multifunctional sensor nodes. Usage of WSANs in automation systems allows replacement of the cables. This reduces the implementation cost and maintenance efforts of the networks. Compared to wired systems, WSANs also come with the advantage of easier and faster deployment, reconfiguration and expansion capabilities, and realisability of applications which are impossible to carry out by using cabled systems. Cost effective replacement of wired systems yields deployment of large number of measurement points, development of self–organised, collaborating and self–healing systems (Çuhac, Yiğitler, Virrankoski & Elmusrati: 1).

Wireless sensor and actuator networks need to retrieve information about the current situation of the environment in order to change the state of the physical world. Environment here can be defined as objects and their properties that are surrounding the network. An important way of having information about the current state of the environment is to obtain visual information retrieved by vision sensors which allow determination of various object properties such as quantity, size, speed, colour, distance and so on. Thus, wireless sensor nodes which are equipped with vision sensors are essential for various applications.

Due to the power limitation of the wireless sensor nodes, vision sensor equipped wireless sensor nodes focus on the acquisition and transmission of the image directly to a central computer in order to maintain low power operation. Since the transmission bandwidth and computation power are quite limited in wireless sensor nodes, they either perform very basic operations on visual data like line tracking or they perform computations with low frame rates (Çuhac 2010: 1).

3.2. The UWASA Node

The UWASA Node is a wireless sensor node designed to realise such a platform that provides fast adaptation and development of various wireless automation applications. This generic platform is achieved by stacking rather small simple slave modules on the main module of the node. The UWASA Node has a modular and stackable hardware architecture represented in Figure 4 (Yiğitler 2010a: 2–4).

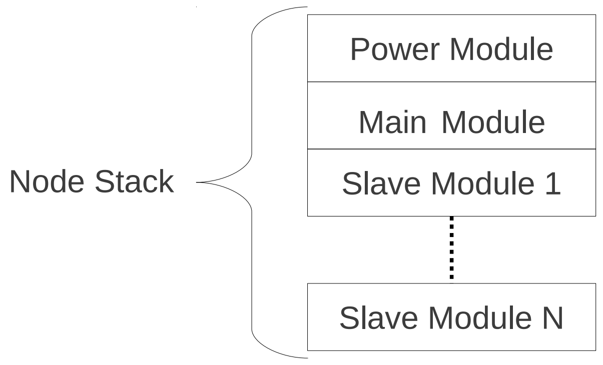


Figure 4. Hardware model of UWASA Node.

The node has two essential modules which are called the Power Module and the Main Module. They provide the fundamental properties and capabilities what makes it a generic wireless sensor node. Those properties and capabilities are wireless communication interface, support for many peripheral interfaces, basic processing and memory, power management and distribution interfaces.

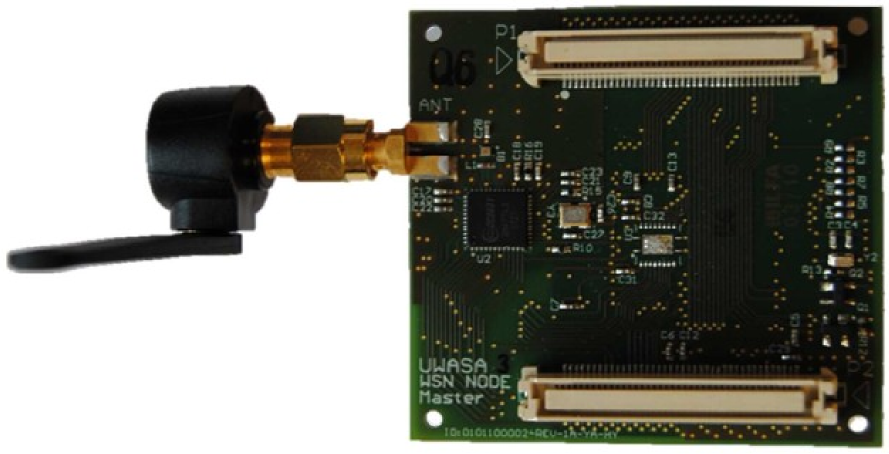


Figure 5. Main module of the UWASA Node. Slave modules can be stacked onto white connectors.

One or more simple slave modules can be added to the hardware stack and they are application dependent custom designs. Signalling and power supply are transferred to these slave modules via hardware stack connector regardless of type and number of the slave modules.

The UWASA Node is designed to support from low power applications like relaying the signal as simple transceiver up to applications that require high processing power and complicated interfacing. The node can either be used without any slave module that simply acts as a low power wireless transceiver or as a device that is equipped with higher amount of resources and slave modules.

3.3. Image processing in wireless sensor nodes

Computers can't learn and reason the events like humans or animals are able to. Therefore raw image data is rarely useful for computers to be able to perform tasks based on visual information. In order to decide the next action to take based on visual information, or do something specific to meet the application demands, image data should be transformed to a level which represents useful information to the system. This transformation is called image processing.

Formally, image processing is defined as a type of signal processing that takes an image as input, and generates an output either as a form of another image or set of useful data and parameters related to input image.

The low power nature of the wireless sensor nodes imposes the limitation to the computation power. Since the images usually contain much larger data than most other forms of information, image processing in wireless sensor nodes need to be limited up to a reasonable level, as well as some easy computing methods may be implemented in order to reduce computation efforts.

An efficient computation reduction is achieved by feature extraction. In some cases, the input of the image processing system may contain very large data so it may be difficult to process or transmit. In such situations the system can take the advantage of selecting the useful data that represents same amount of information which is needed to compute the output. Hence the input data firstly can be transformed to a reduced form, from which the output can accurately be determined. This transformation of the input data is called feature extraction.

Important point in feature extraction is the accuracy of the extracted data. Extracted features must be collected very carefully so that it must still represent the relevant information existing in the input data.

An example of a feature extraction which is applied in this work is given in Figure 6.

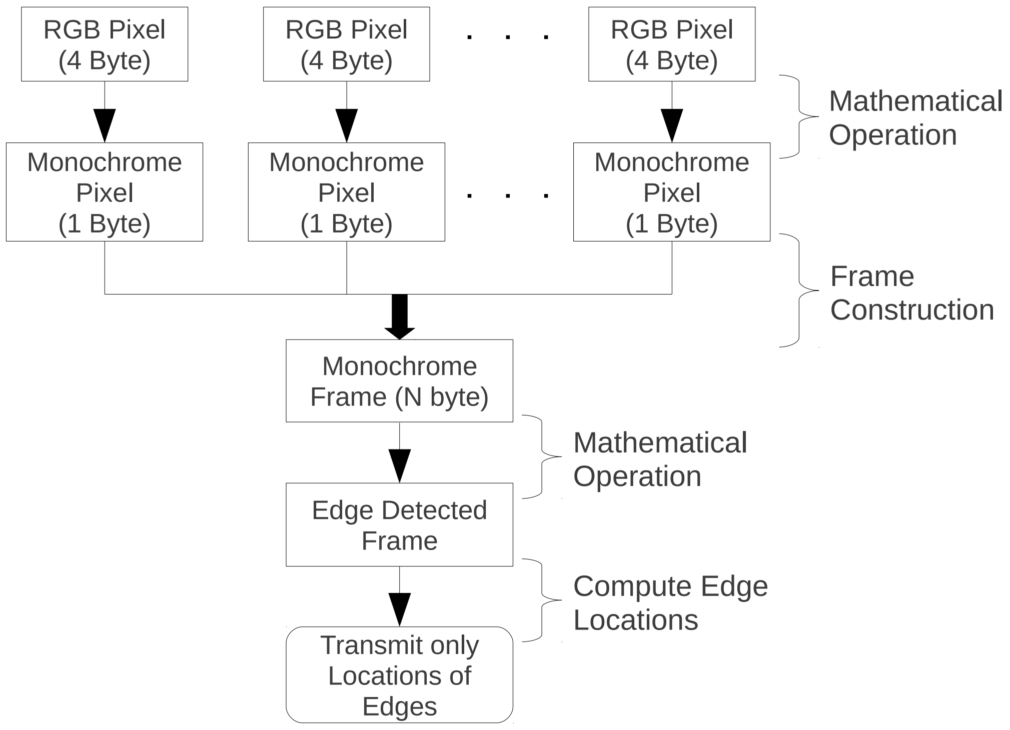


Figure 6. An example of a feature extraction that is applied in this work.

In this algorithm as soon as the data in RGB format is acquired, it is reduced to one quarter sized monochrome format before being placed in the memory. As given in the definition of the feature extraction above, the input data is reduced. Again, in definition it is stated that after the feature extraction, desired results must be obtainable using extracted features. An important point to mention here is that the edge detection is performed over a monochrome image. Since the purpose is to perform edge detection, monochrome image still represents almost the same information represented by the RGB image. Therefore it is possible to compute edges, and that concludes all the criteria of the feature extraction concept are verified.

The transmission part located in the end of Figure 6 is not performed in this work because this work is a proof of in–node image processing thus the transmission is not involved.

Another feature extraction would be easily performed in the last step. After computing the edge locations in the frame, it may often be easier to transmit only the coordinates of the pixels that represent edges. Furthermore, if this would be a part of a continuous image acquisition loop, transmitted data could only be the difference from the previous image. By doing so, data transmission related to edge detection could be reduced to tiny amounts while still advertising the results over the network.

4. HARDWARE

Three main hardware blocks of this design are a development kit with an ARM processor, a camera board, a test board.

The camera board is a PCB that contains a vision sensor and a connector. The vision sensor located on the camera board acquires the image data and delivers it over the pins located on the connector. The test board acts as an interface between the camera board and the processor. This chapter describes the structures of the designed hardware blocks, gives an overview of hardware blocks' functionalities and capabilities, and explains how they are related to each other.

4.1. Development kit

In this work Olimex LPC–2378STK development board is used as a representation of UWASA Node since they both have the same microcontroller. This work is a proof of concept, therefore instead of producing a complete slave module for UWASA Node, the external connections of the development board is used to communicate with the prototype test board in order to ease the hardware design and production. Development board contains many peripheral interfaces but in order to comply with the test conditions, only the pins which have direct connection to the processor are used.

The development kit and the blocks which are used in this work are shown in Figure 7.

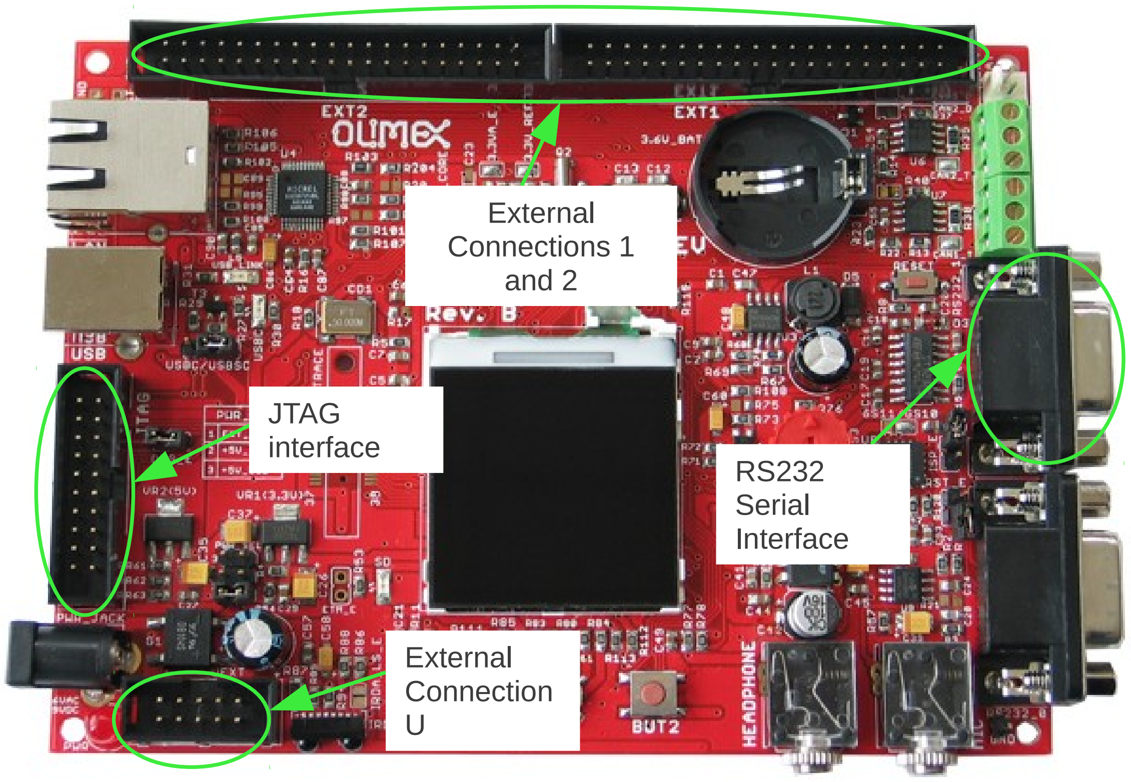


Figure 7. LPC–2378STK development board and used interfaces. (Olimex 2010)

Development kit is either powered by an external power source, JTAG interface or USB. Like the UWASA Node, development kit is also programmed using JTAG interface so it was used as a power source too. External Connections 1, 2 and U are the connections which introduce the development kit as UWASA Node to the test board. RS232 serial interface is not needed for the operation but it is used to display the computed results on the computer screen.

In order to connect the computer to JTAG interface for programming and debugging, USB to JTAG adapter is used. JTAG adapter collaborates with the software development environment to allow stepwise code debugging on hardware.

4.2. Camera board

There are already developed camera platforms with low power microcontroller interfaces. Though those devices are low power, the power consumption of the image acquisition in wireless sensor networks is more or less same as the transmission power (Culuricello 2006: 39). CMUcam3 is one of those platforms which has open software. It provides basic vision capabilities to small embedded systems in the form of an intelligent sensor. CMUcam3 complements the low cost hardware platform by providing a flexible and easy to use open source development environment which makes it a good candidate to work with. Additionally, it is based on LPC2106 microcontroller which belongs to the same family with the UWASA Node’s LPC2378 microcontroller.

CMUcam3 basically consists of two different boards connected to each other: the camera board and the main board. Those two boards are connected to each other with standard 32–pin 0.1 inch headers. The processor, power connections and the FIFO chip of the CMUcam3 are located on the main board while the camera board only consists of a vision sensor and a header connected to sensors pins. Figure 8 shows the complete CMUcam3 structure.

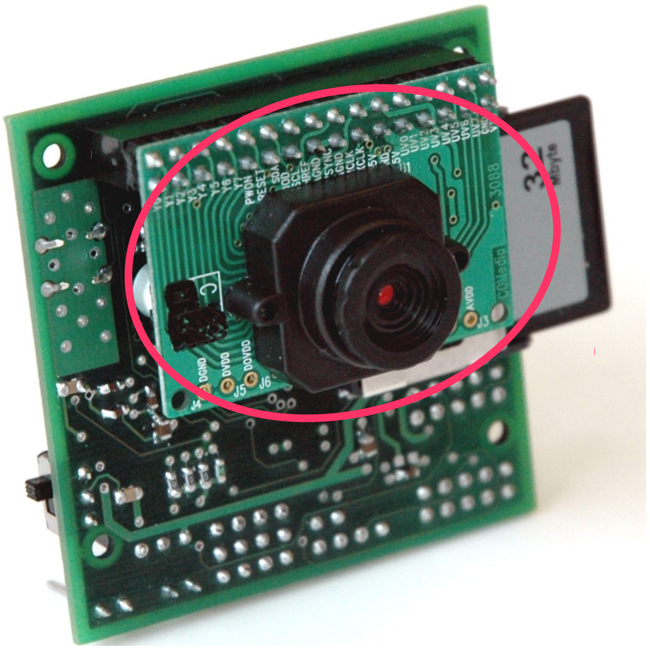


Figure 8. CMUcam3. Camera board is on the front (CMUcam3 2011).

In this design, only the camera board of the CMUcam3 is used as vision sensor. This architecture aims to enable easy replacement of the vision sensor depending on the application requirements. Since the behaviour of this slave module reflects all of the hardware related features of CMUcam3, it may also be possible to substitute the camera board with another one having different specifications.

The camera board of CMUcam3 is a portable PCB circuit that integrates some passive components, OV6620 vision sensor, and a header. Header represents some of the vision sensor pins to external devices.

The pins available on the camera board header are given in Table 1 with their functionalities.

Table 1. Available camera board pins and their functions.

Pin	Function	Pin	Function
1–8	Digital Output Y Bus	17	Analogue Ground
9	Power Down Mode	18	Pixel Clock
10	Reset	19	External Clock
11	I2C Serial Data	20	+5 V DC
12	Odd Field Flag	21	Analogue Ground
13	Serial Clock	22	+5 V DC
14	Horizontal Reference	23–30	Digital Output UV Bus
15	Analogue Ground	31	Common Ground
16	Vertical Sync	32	Video Out (75 Ω)

The 8–bit data output pins are pin 1 through pin 8. Pins 23 through 30 are active only when the vision sensor is used in 16–bit mode so they aren't used in this design. Pins 9 and 10 are connected directly to a GPIO pin of the processor through the test board to power down or reset the camera respectively. Pins 11 and 13 represent the SCCB bus that is used to configure camera options and it operates in a similar way to I2C standard. Pins 20 and 22 are 5 V DC supply voltage pins of the vision sensor. Since the UWASA Nodes power module doesn't provide 5 V, a DC to DC conversion from 3.3 V is necessary. This DC to DC conversion is discussed later in section 4.4.5 DC to DC converter.

The rest of the pins aren't needed and not used in this design except horizontal reference, vertical sync, and pixel clock. Those three pins carry the vision sensor output signals which are vital for timing, synchronisation, and acquisition of the image data.

4.3. Specifications of the vision sensor

The vision sensor OV6620 which is used in the design is able to output images at a maximum resolution of 352 x 288 pixels up to 60 fps. It can be configured via SCCB interface to output in 8–bit or 16–bit, RGB or YCbCr colour modes. The maximum power consumption of the camera is 80 mW and operates at 5 V DC. It is not a sophisticated vision sensor but since this work is focused on limited image processing, it is enough to show the proof of the concept.

4.3.1. Resolution

The Omnivision OV6620 vision sensor captures the images with an array of 356 x 292 photosensors. In this vision sensor, each pixel is represented by four values: B,G,R,G.

Table 2. Semiconductor array of the vision sensor.

Row \ Col	**1**	**2**	**3**	**4**	**...**	**353**	**354**	**355**	**356**
1	B_{11}	G_{12}	B_{13}	G_{14}		B	G	B	G
2	G_{21}	R_{22}	G_{23}	R_{24}		G	R	G	R
3	B_{31}	G_{32}	B_{33}	G_{34}		B	G	B	G
4	G_{41}	R_{42}	G_{43}	R_{44}		G	R	G	R
5									
...									
289	B	G	B	G		B	G	B	G
290	G	R	G	R		G	R	G	R
291	B	G	B	G		B	G	B	G
292	G	R	G	R		G	R	G	R

As mentioned before, the maximum resolution that can be output is 352 x 288 but this resolution is achieved by generating pixels that share the same photosensor value. To make it clear, the first pixel is generated using elements B_{11}, G_{12}, R_{22}, G_{21} and the second pixel is generated using elements B_{13}, G_{12}, R_{22}, G_{23}. Among those sets of elements G_{12}, R_{22} are commonly used to generate two different pixels. In the elements of the third pixel, there will be common elements with the second one and so on. Information rate in the maximum resolution image can be determined by the ratio of unique information versus total information.

$$\alpha = \frac{U}{T} = 356 \times \frac{292}{352 \times 288 \times 4} \approx 0.5 \tag{20}$$

where,

α = Information ratio

U = Number of unique elements used to generate the image

T = Number of total elements used to generate the image

This result shows that in fact the vision sensor is not capable of providing 100% informative image at a resolution of 352 x 288 pixels. Because of that, instead of setting the vision sensor to generate output at maximum resolution, a mode which has lower resolution with the information rate of 100% is used in this design.

4.3.2. SCCB interface

The registers of the OV6620 vision sensor are configured via SCCB (Serial Camera Control Bus) interface. Those registers keep the values for various camera settings as long as the camera is continuously powered.

SCCB is a two wired serial interface that operates very similar to I2C standard. It supports up to 400 kbps serial transfer rate using 7–bit address and data transfer protocol. Within each byte, the MSB is transferred first and the last bit of the address byte indicates whether the operation is read or write. Vision sensor is always a slave device.

Write operation in SCCB bus is initiated by firstly transmitting a start condition. After the start condition, slave device is aware of an ongoing communication. Any write operation consists of three bytes. The first byte is the write address of the device to be accessed. It is a fixed hexadecimal value (0xC0) for the camera board used in this design. The second byte is the address byte of the register, and the last byte is the value to be set.

Start	Write Address of the Slave (0xC0)	ACK	Register Address	ACK	Register Value	ACK	Stop

Figure 9. SCCB write operation.

After the register value is sent, a stop condition occurs to inform the slave device that the communication is terminated.

Just like the write operation, the read operation is also initiated when a start condition occurs. However, read operation consists of two bytes. The first byte is a fixed value (0xC1) which is the read address of the slave device. After the master sends the read address of the slave, slave device outputs the value of the last written register to the bus.

Start	Read Address of the Slave (0xC1)	ACK	Register Value	Stop

Figure 10. SCCB read operation.

Similar to the write operation, a stop condition occurs to inform the master device that the communication is terminated.

4.3.3. Frame rate

OV6620 vision sensor can output images up to 60 frames per second. Frame rate is independent of the image size and is configured via SCCB registers at addresses 0x2A and 0x2B. 0x2A contains the frame rate adjust enable bit and the MSB of the frame rate value while 0x2B register contains least significant bits of the frame rate value. After the frame rate adjustment is enabled, 512 different levels can be selected. Frame rate varies from 0.21% up to 109%. For example, in case frame rate needs to be adjusted to 10 fps, first the register values must be calculated as follows:

$$\frac{10\,fps}{60\,fps} = \frac{x\,\%}{109\,\%} \;\rightarrow\; x = 18.16 \qquad (21)$$

Now the register value should be set to:

$$\frac{18.16}{0.21} \approx 86 \tag{22}$$

which corresponds to 0x56 in hexadecimal.

4.3.4. Data format

Available data formats for OV6620 vision sensor are the combinations of YCbCr or RGB colour modes, and 16–bit or 8–bit data modes. In the applications of this design, RGB with 8–bit mode is used.

RGB mode is selected by setting the fourth bit of the register at address 0x12, and 8–bit mode is selected by setting fifth bit of the register at address 0x13 using the SCCB bus.

4.3.5. Timing

Timing synchronisation is done by using PCLK, HREF and VSYNC signals. Those signals indicate the data bus validity, row output time duration, and start of a new frame respectively.

Timing diagram of the vision sensor output signals for one row duration are given in Figure 11.

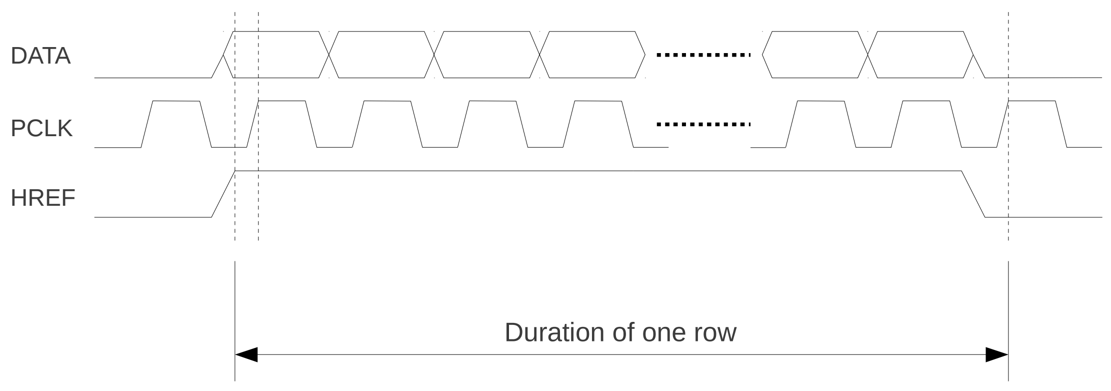


Figure 11. Timing diagram of the vision sensor output signals for a row.

PCLK signal is a clock signal that's why it alters its logic state continuously regardless of the data existence on the bus. For that reason, it is impossible to determine in which clocks the data is coming by only this signal. In order to clear this situation, horizontal reference signal stays active during the image data output time span. In other words, HREF signal stays active only when there is a meaningful data on the bus. HREF is an indication for a complete image row duration.

The vision sensor starts to output the image with the first pixel of the first row, continues until the end of that row, and then goes on with the second row. This process repeats until the last row is output. After each image frame, a VSYNC signal indicating a start of a new frame is asserted for synchronisation. The first HREF after the VSYNC signal marks the first row of that image. Unlike HREF, VSYNC signal does not maintain logic high level during the entire frame. How those two signals are coupled with each other is given in Figure 12 below.

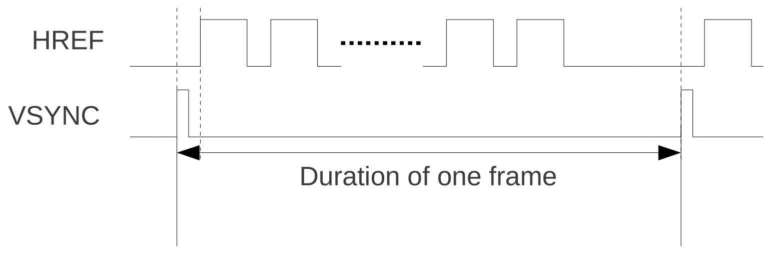


Figure 12. HREF and VSYNC signals for one frame. VSYNC marks the start of each frame.

4.3.6. Auto–adjustment options

Many properties of image acquisition can be adjusted automatically by the camera rather than manually setting them. OV6620 vision sensor is able to iteratively find the optimum values that results the best image quality and highest SNR. Once the camera is powered up, the internal circuitry calculates those values and set the corresponding registers automatically.

Some significant properties which can be auto–adjusted are listed in Table 3.

Table 3. Automatically adjustable properties of the vision sensor.

Auto Adjustable Properties
Red, Green, Blue channel gains
Saturation Control
Contrast Control
Brightness control
Sharpness control
White balance background
Exposure control

4.3.7. Camera connection to the test board

In order to ensure physical flexibility, the camera board is connected to the test board via IDC cable so that it is easy to rotate by hand without moving the whole hardware.

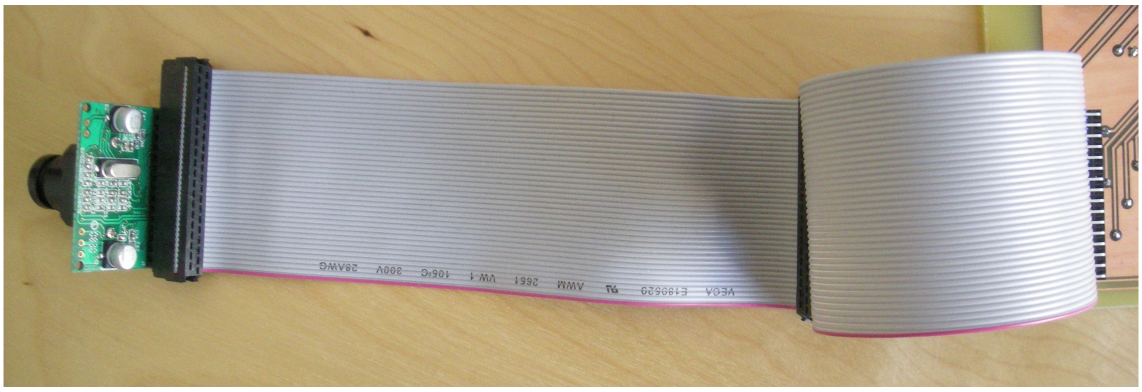


Figure 13. Camera board is connected to the test board via IDC cable.

Vision sensor has a lens in front of it which focuses the image on the semiconductor array. Depending on the interested distance, this lens must be adjusted manually for the best quality.

4.4. Test board

The test board is the interface between the camera board and the processor. Together with the camera board, it represents the prototype version of the slave camera module for UWASA Node.

The hardware has firstly been designed in a schematic level on the PC. The schematic drawings of each hardware block can be found in the appendices. Upon the completion of the schematic design, PCB layout was designed and routed. The three dimensional view of the test board PCB is represented in Figure 14.

After the PCB design, the prototype circuit has been produced and tested in both electrical and physical level.

The hardware elements of the test board can be divided into groups as FIFO, SRAM, DC to DC converter, octal bus transceiver, logic components, and passive components.

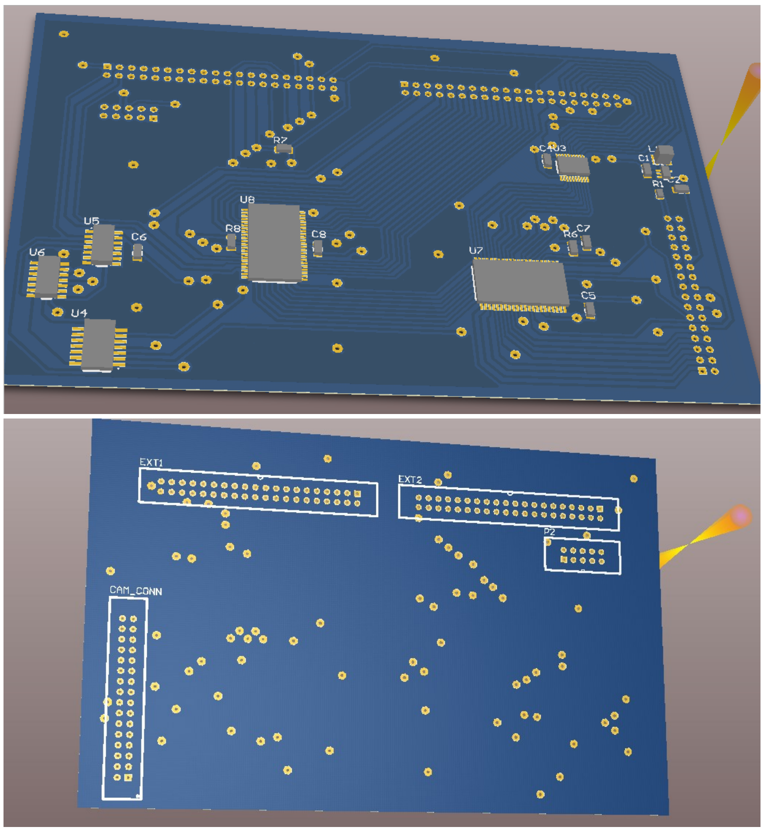


Figure 14. Front and back of the test board in 3D view.

4.4.1. FIFO concept

FIFO, meaning First–In–First–Out in computing and electronic design, is a concept of organising data transfer efficiently between the data source and destination having different speeds.

The data output rate of the source and the data handling capability of the destination may sometimes be different. In case the source outputs the data faster than the destination can handle in average, theoretically the queue would go to infinity. But in practice, since the memories are limited, such a situation results in data loss. If the source outputs the data slower than the destination can handle, then there will be a limited queue in the system.

FIFO buffers the incoming data from the source system. The destination system which is capable of handling data at a faster average rate than the source system outputs, waits for a certain time so that a considerable amount of data is accumulated in the buffer, then empties the buffer quickly. This way the destination system doesn't need to handle the data very frequently since it has the capability of handling larger amount of data at once.

4.4.2. Advantage of using FIFO

In section 2.2, the average number of units in the system, in other words, the length of the queue is given by:

$$L = \sum_{n=0}^{\infty} n P_n \tag{23}$$

Using equation 11:

$$= \sum_{n=0}^{\infty} n\left(\frac{\lambda}{\mu}\right)^n\left(1-\frac{\lambda}{\mu}\right) \tag{24}$$

$$= \frac{\lambda}{\mu-\lambda} \tag{25}$$

If the camera was directly connected to the processor, to be able to handle all the incoming data, there had to be no queue in front of the processor input which means L had to be equal to 0. If there was any data bit which wasn't handled by the processor at a time, and there came another bit, the one that hadn't been handled until that time would disappear from the data bus, and that would lead to inconsistency in the system. Achieving zero length in the queue would be possible when there is an infinitely great handling rate μ.

Those results prove that between the camera and the processor, there must be some memory to buffer the data because the processor can't handle the data on the bus with infinitely small time intervals. The write clock signal from the camera is directly connected to the FIFO buffer, similarly the read clock from the processor is also connected to the FIFO buffer and they operate independently from each other. This way it is ensured that all data that comes inside the buffer will stay there until the processor reads it.

4.4.3. FIFO chip

The FIFO chip used in this design has 512 kB x 8–bit memory and completely independently operating input and output ports each having 8–bit data widths. Output ports can operate up to 80 MHz and are supported by built–in circuitry which provides pointer reset, data skipping and some more useful functions that make it an easy to use memory device. Power supply is rated at 3.3 V and the chip has around 55 mA of current consumption at given voltage.

AL440B has 44 pins in total but only the ones that have importance in this design are given in Table 4 below.

Table 4. Significant pins of AL440B FIFO chip and their functionalities.

Pin	Function
DI[0..7], DO[0..7]	Data Input Bus, Data Output Bus
WE, IE, RE, OE	Write Enable, Input Enable, Read Enable, Output Enable
WCK, RCK	Write Clock, Read Clock
WRST, RRST	Write Reset, Read Reset
SDA, SCL	Serial Data, Serial Clock
SDAEN	Serial Data Enable
PLRTY	Polarity of the control signals
RESET	Reset FIFO chip.
IRDY, ORDY	Input Ready, Output Ready
VCC, GND	Supply Pin, Ground Pin

FIFO chip has no address bus. Memory access for both write and read operations is conducted by write and read pointers. Those pointers are incremented by clock signals WCK and RCK when WE and RE signals are active respectively. Write and read pointers are always incremented. When pointer reaches to the end of the memory it restarts from the first address. Another way to set the pointers to the initial positions is using WRST and RRST signals.

Data input bus of the FIFO is directly connected to the camera data output bus, and WCK input is connected the PCLK signal of the camera board. Hence, the data that comes from the vision sensor is directly written to FIFO chip. The hardware structure that connects the camera board to the FIFO is given in Figure 15.

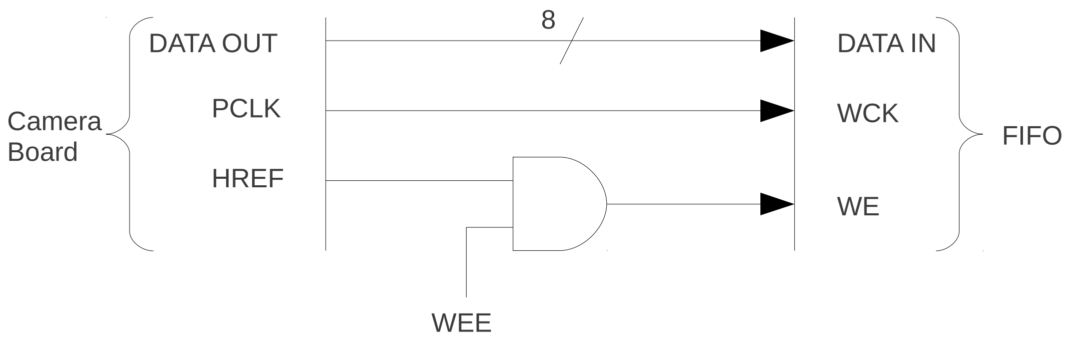


Figure 15. Connection of FIFO to camera board.

Here WEE signal allows selective data reading. Details about this signal is discussed further in section 4.4.7.

WE and IE signals have different functionalities. If both IE and WE are active, normal read operation is done and write pointer is always increased with WCK. In the presence of WE but not IE, write pointer is increased but the data on the input bus is not written to internal registers. This is very useful feature for write skipping. When WE signal is not active, regardless of IE, both data input and WCK are disabled. Similar relation applies to OE and RE signals. When the output is not enabled but RE signal is active, read pointer is increased for data skipping.

4.4.4. SRAM chip

SRAM is a type of semiconductor memory which is capable of keeping the register values continuously as long as it is powered. Unlike DRAM, SRAM doesn't need to be refreshed within certain amount of time intervals to be able to keep the values in registers.

The embedded applications that work with images occupy much more memory than ordinary embedded applications because the image must somehow be stored in a memory. For that reason this hardware design consists of an SRAM to store the image, to process the image, and to keep the resulting image after processing.

SRAM used on the test board is IS61LV5128AL chip. Main features of this chip are its high speed access time which is as low as 10 ns, power down option, fully static operation without any clock, 3.3 V supply voltage, and 80 mA typical operating current.

The most important feature of this chip is its access time which is only 10 ns. The access time from the processor side has a high scalability since it is done by external memory controller. EMC can be configured by the internal registers of the processor, and can be precisely adjusted to achieve the most efficient data handling in both reading and writing periods.

Pins of the SRAM and their functions are given in Table 5 below.

Table 5: Connections available on SRAM chip.

Pin	Function
A[0..18]	Address Bus
CE, OE, WE	Chip Enable, Output Enable, Write Enable
IO[0..7]	Bidirectional Ports
VDD, GND	Supply, Ground

The image that is obtained from the vision sensor has a size of 176 x 144 pixels. In raw image data format each pixel is represented by four bytes. Hence the size of an image on the memory is:

$$4 \times 176 \times 144 = 101376\ B = 99\ kB \tag{26}$$

The EMC can address up to two adjacent memories at a size of 64 kB, so the image can be stored on the 128 kB reserved location on the SRAM. Since the SRAM chip has a size of 512 kB, four different forms of images can be stored on it and processed. The allocation of those extended memories is done by using two more GPIO pins connected to two most significant address pins. The structure that is capable of doing such an addressing is given below in Figure 16.

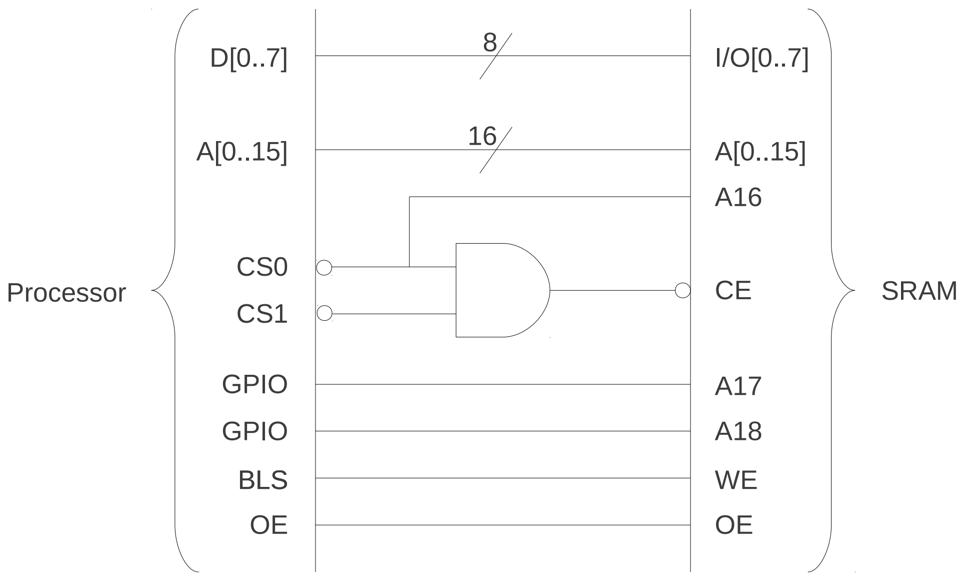


Figure 16. The connection between SRAM and the processor. Here addressable memory is quadrupled by using GPIO pins.

4.4.5. DC to DC converter

The power module of the UWASA Node doesn't have 5 V DC supply. The whole node operates with lower DC voltages, so does the slave modules. In order to generate a 5 V DC supply for the camera board, DC to DC step converter has been used on the test circuit.

The logic behind the DC to DC step converter is that is uses an inductor, an integrated switching circuit, and two capacitors to filter both the input and output. Inductor behaves like a short circuit in DC circuits, and like an open circuit for infinitely high frequency. The current equation for an RL circuit is given by (Serway & Jewett 2010: 931):

$$I=\frac{\varepsilon}{R}\mathrm{e}^{-t/\tau}=I_i e^{-t/\tau} \tag{27}$$

Here:

I = Current

ε = Electromotive Force

R = Resistance

t = Time

τ = Time constant of the circuit

Equation 27 shows that, for very short t values compared to τ, an inductor has a tendency to keep the flowing current constant through itself. In DC to DC step converter, this small t is achieved by the switching circuit. Closing the switch makes the current flow through the inductor and when the switch is opened, inductor behaves as a current source. The rate of switching and the duty cycle determines the output voltage of the DC to DC converter. The application circuit of this conversion is given below in Figure 17.

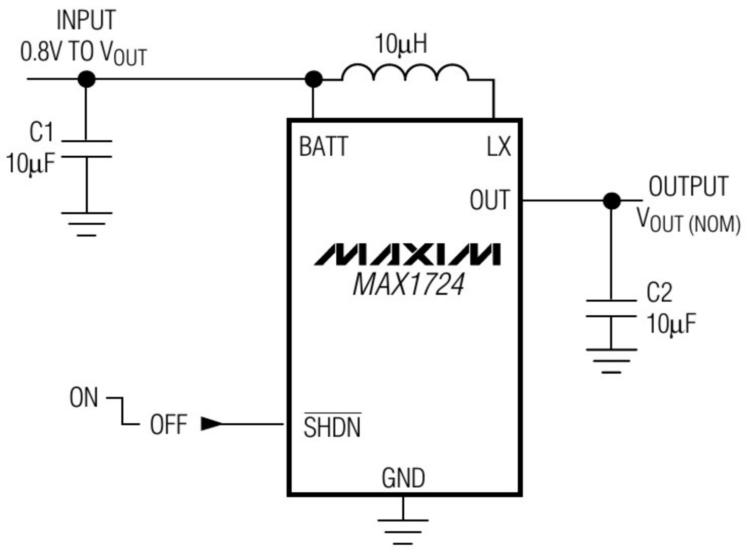


Figure 17. Application circuit of the DC to DC converter (Maxim 2011).

4.4.6. Octal bus transceiver

Octal bus transceiver on the test board has been placed to isolate the EMC from the common 8–bit data bus which is accessed by both FIFO and SRAM. Like SRAM, this chip also has a very high access time as low as 6 ns. It can totally isolate the EMC from the shared data bus or set the data direction either from EMC or to EMC.

When OE is not active it isolates the two sides of the bus but when active, data direction can be switched regarding to the logic state of the DIR pin located on the chip.

The aim of using such a property was to enable glue–less data transfer from FIFO to SRAM but due to some limitations that sort of transfer is not performed in this work.

4.4.7. Logic components

Logic components on the test board have essential importance. There are three logic chips which are inverter, AND gate, and a D–type Flip–Flop.

Inverter is used to alter the logic levels of some signals in order to comply with the active low or active high properties of the signal inputs, or to comply the logic design necessities.

AND gate has an important role on the communication interface between the EMC and the SRAM. EMC has two memory bank selection signals, namely CS0 and CS1. 16 address pins A[0..15] of the EMC are capable of addressing 64 kB of memory at most. When CS0 is active, the lower 64 kB memory bank is selected and when CS1 is active the higher 64 kB memory bank is selected. The following logical structure in Figure 18 is designed in order to address adjacent memory banks on the SRAM.

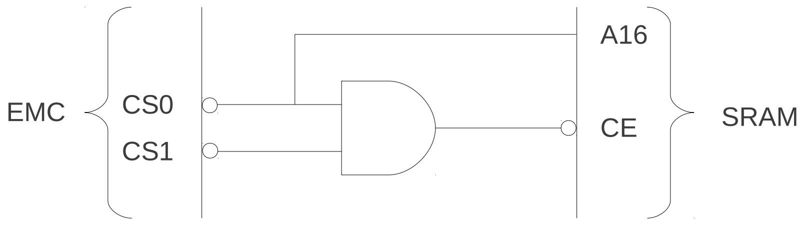


Figure 18. Adjacent memory bank selection logic

Another logic component used in the hardware design is D–type flip–flop which has an important role in the image acquisition circuit. It makes the incoming data to be parsed in the forms of frames on the FIFO chip. The hardware design of how the image is a–synchronised using a D–type flip–flop is represented in Figure 19.

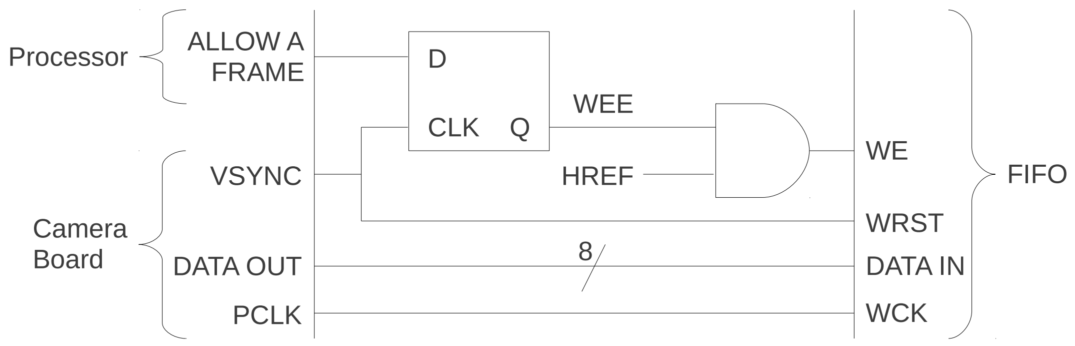


Figure 19. Usage of the D–type flip–flop in frame acquisition.

Unless the processor does not activate ALLOW A FRAME signal, flip–flop keeps the WEE signal deactivated thus FIFO WE signal is never enabled. In this situation no data is written to FIFO. When this signal is activated and kept at high level, with the first VSYNC signal WEE is activated during the active periods of HREF signal, data in the bus will be continuously written to the FIFO register with each PCLK signal. VSYNC signal is also connected to WRST pin so that each frame is always written by starting from the first memory location of the FIFO. This makes it much more easy to handle the data located inside the FIFO chip. Continuous frame transfer can easily be done by keeping the ALLOW A FRAME signal active.

4.4.8. Passive components

Passive components used in the circuit are some capacitors, resistors, and an inductor. Many of the capacitors are implemented in order to increase the noise immunity. Noise filtering capacitors are connected parallel to the signal to the ground. All the resistors except one are used as a pull up resistor to ensure that the signal stays high unless desired to be low. This prevents the signals from floating when they are not driven by a high or low logic voltage. One resistor with a value of 0 Ω is used to isolate the ground of the test board from the ground of the camera board. The inductor is used to generate 5 V supply voltage for the camera board.

5. SOFTWARE

The software in this design defines the interaction and communication rules between the camera, test board, and the processor. Another role of the software is to establish communication with a computer in order to display the graphical results of the operations as a bmp file. The software is written in C language and doesn't run on any operating system.

5.1. Overview of the software

The main function algorithm starts by initialising the processor and interrupts. Then it sets the functions of the used pins and their initial states. The LPC2378 processor has four ports and almost all pins of each port can be assigned up to four different pin functionalities. For example, pin 21 that belongs to port 1 can be configured to be operating in GPIO, USB, PWM, or SPI modes. In order to comply with the hardware properties, the necessary configuration is done before doing any operations.

After the initialisation of the pins, the software powers up the camera and configures the it to work in 8–bit RGB mode. Then the algorithm waits for some time to allow the camera to stabilise with its internal auto adjustment circuitry. Following the stabilisation, ALLOW A FRAME signal is set active so the FIFO grabs an image from the camera. After waiting for at least 1 frame duration, all the inputs to FIFO chip is disabled and the image is stored inside the chip. At that point, before reading the image, an image file header is generated. As soon as the file header is sent to computer via serial terminal, next step is to read the partial data and process it. Necessary signals are then configured to enable the read operation. Details about reading and processing the image are described further in section 5.4.

Finally the resultant image is sent to the computer and the software enters an infinite loop. Flowchart of the main function is represented in Figure 20.

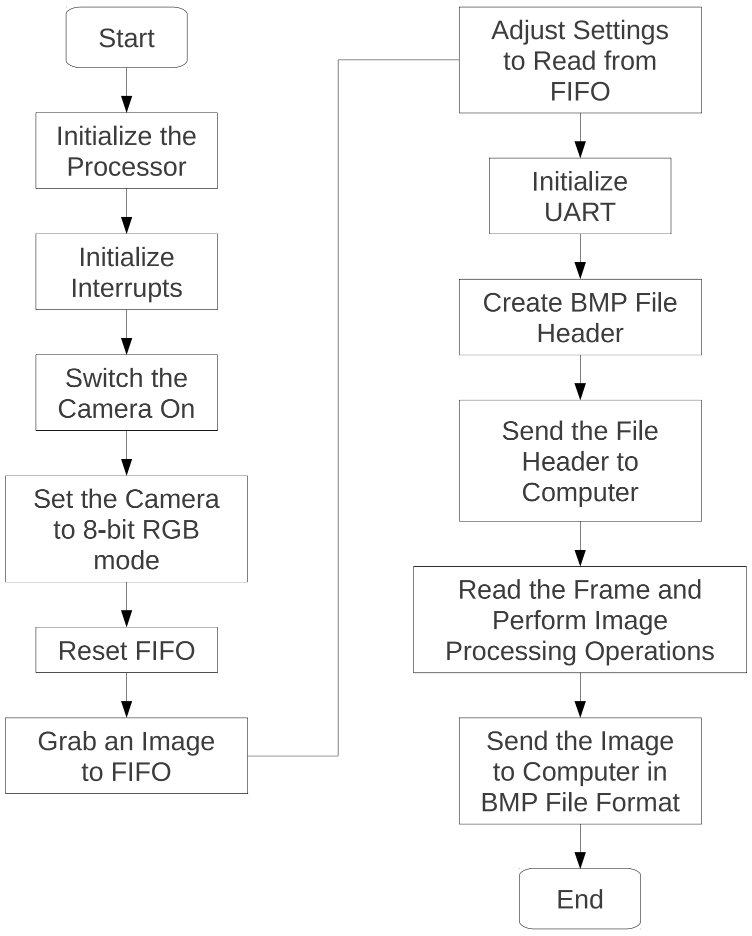

Start
Initialize the Processor
Initialize Interrupts
Switch the Camera On
Set the Camera to 8-bit RGB mode
Reset FIFO
Grab an Image to FIFO
Adjust Settings to Read from FIFO
Initialize UART
Create BMP File Header
Send the File Header to Computer
Read the Frame and Perform Image Processing Operations
Send the Image to Computer in BMP File Format
End

Figure 20. Flowchart of the main function.

5.2. Camera settings

Camera settings are configured using I2C peripheral. SCCB bus of the camera operates in a very similar way to I2C standard, so suitable processor pins are assigned to operate in I2C mode.

In order to set the value of any camera configuration register, firstly the address of that register is assigned to a char sized variable. This variable is the input parameter of the register setting function. After the register setting function transmits its input parameter as the register address, it checks for an acknowledgement. Then in the same way the register value is sent without transmitting a stop condition in the middle. If an error occurs during this process, the interrupt handler terminates all operations. After a register is set, the processor asks the camera to read back the value it has received in order to ensure the configuration is done properly.

5.3. Creating a bmp file

File format bmp is a commonly used image file format that stores digital images as a bitmap with its various properties like width, height, colour depth, and resolution etc. All those properties and many others take place in the bmp file header.

In order to write the image as a file on the hard disk of the computer, the file header creator function in the software automatically generates the parameters of the bmp file header. The structure of the bmp file format is given in Table 6 below (Frontier 2011).

Table 6. bmp file format. File header excludes the colour map given in the last row.

Offset	Size	Contents	Description
00	02	“BM”	Microsoft’s bmp ID word
02	04	Varies	Size in bytes of the file
06	04	00, 00	Reserved
10	04	Varies	Offset in file where image starts
14	04	40	Size of bitmap header
18	04	Varies	Width in pixels
22	04	Varies	Height in pixels
26	02	1	Number of image planes (only one)
28	02	Varies	Bits per pixel (1,4,8, or 24)
30	04	Varies	Compression type
34	04	Varies	Size of compressed image (or zero)
38	04	Varies	Horizontal Res. in pixels/meter
42	04	Varies	Vertical Res. in pixels/meter
46	04	Varies	Number of colours used
50	04	Varies	Number of 'important' colours
54	04	Varies	Colour Map

File header creator function creates an array according to Table 6 and then passes this array to another function which sends it to the computer over the serial interface. Since it is a simple matrix construction regarding to the table above, the flowchart is not represented here.

5.4. Reading from the FIFO

After a stable image is stored in FIFO, all of its input signals are disabled so the image is never overwritten during a read operation. As described before in section 4.4.7 the image data always starts from the first location on the FIFO, because within each incoming frame, a write reset is applied to FIFO.

After the necessary settings are done, read clock signal is set to high level so that the data is present on the bus. In that moment, pixel data variable in the processor is assigned to the 8–bit value on the bus, and read clock is lowered back. When this operation is performed, FIFO automatically increments its read pointer so the next time read clock goes high, next value will be on the bus and so on. This process repeats itself up to desired number of pixels is obtained.

5.4.1. Determining the first pixel of the frame

As stated before, the first pixel of the frame should always be located in the first memory location of the FIFO, but in the datasheet of the camera it is stated that in 8–bit RGB mode, the first row of the frame is always unstable. This puts a challenge of finding which byte represents the first byte of the frame. At that point live debug option of the compiler enabled to figure out that the byte sequence before the first byte of the frame follows this order in hexadecimal:

?	?	0x10	?	0x10	1st Byte	2nd Byte

Figure 21. The unstable output sequence before a frame starts.

Here the question marks stand for a random value. Noticing that two 0x10 values follow each other with a random value between them before the frame starts, it was possible to parse the data correctly. This sequence may occur at different locations of the FIFO memory every time, thus a sequence detector function had to be written. The situation introduces instability to the system and is not mentioned in the vision sensor datasheet. Possible reason is that VSYNC signal is not very accurate in timing.

5.4.2. Reading and placing pixels in correct order to form a frame

In 8–bit RGB mode, the vision sensor outputs the data in the following order:

Table 7. 8–bit RGB mode output sequence from the vision sensor.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**...**
G	B	G	R	G	B	G	R	G	B	...

On the other hand bmp file format has a pixel order as:

Table 8. Distribution of the colours in bmp file.

R\C	**1**	**2**	**3**	**4**	**5**	**6**	**...**
1	R	G	B	R	G	B	
2	R	G	B	R	G	B	
3	R	G	B	R	G	B	
4	R	G	B	R	G	B	
5	R	G	B	R	G	B	
6	R	G	B	R	G	B	
...							...

Vision sensor output for one pixel consists of four bytes as shown in Table 7, but bmp file has three bytes in different order. The software firstly takes the four bytes from the camera and right shifts the green values once. Right shifting means division by two for binary numbers. After that, adding them together will result in the average value of those green pixels. Placing the red value first, computed green value second and blue value last, there exists a three byte pixel in correct order. Repeating this operation for every four bytes and joining resultant values sequentially, after a certain amount of times, it forms one row of a frame. Similarly, after successive row computations the colour matrix is completed.

5.5. Processing the image

There are countless existing image processing algorithms like image segmentation, face detection etc. for different application purposes. However, basically there are some very fundamental operations performed on the images in order to obtain those algorithms. One of them is Sobel operator which has a significant importance in edge detection algorithms. Sobel operator works on a monochrome image. It finds the gradient values around each pixel, and generates an output image.

Explanation of the Sobel Operator concept, how it is applied in this design, and how to compute edges inside the image are discussed in this part.

5.5.1. Transformation to monochrome

The processor reads the image from the FIFO in RGB format. In order to apply edge detection algorithms to the image, the image needs to be transformed to monochrome format. Monochrome image is often represented with Y since it is the first component of YCbCr colour space. The transformation to the monochrome from RGB colour space is formally defined as:

$$Y = 0.2125 \times R + 0.7154 \times G + 0.0721 \times B \tag{28}$$

Equation 28 shows that the green component has the most contribution to the monochrome image because the human eye is much more sensitive to green than the other colours.

Since the aim of this work is to test image processing feasibility within the context of limited image processing, instead of following the formal transformation equation, software uses an approximation which saves computation power.

The approximation to the formal transformation applied in the software follows that:

$$Y = \frac{1}{4} \times R + \frac{5}{8} \times G + \frac{1}{8} \times B \tag{29}$$

It should be noted that the coefficients of the colour components are very easy to obtain by using simple bitwise right shift operation and addition. Of course this method introduces some error but on the other hand it is a very easy and fast way compared to having multiplication with fractional numbers and summing them for all the pixels of the image.

5.5.2. Gradient calculation

Once the monochrome image is formed, Sobel operator can be applied on it. Definition, usage and explanation of Sobel operator has been described in section 2.4 before. Again, to reduce computational power, some modifications are applied to this original method and explained.

The Sobel operator has only +2, +1, 0, -1, and -2 coefficients in the matrix. Among those values, only +2 and -2 needs multiplication operation but since they can be obtained with bitwise left shift operation, just summation after left shifting solves the computation in the kernel.

Convolution in two dimension for image processing has been described in section 2.3. Same two dimensional convolution operations are performed for computation of both the horizontal and the vertical gradients. Once the G_x and the G_y orthogonal gradients are obtained as a vector, the magnitude of the vectorial sum must be calculated to find total gradient value.

At this point a problem arises with the computation of big gradient values. The values of the pixels are represented by 256 levels ranging from 0x00 up to 0xFF values in hexadecimal notation. In order to compute the magnitude, the square of those two orthogonal gradient values must be calculated and added together, then the square root operator must be applied.

Assuming that both horizontal and vertical convolution kernels computed the values of 0xFF, taking the second power of 255 and adding same two numbers results in 130050. This number can't be represented by char or short type of variable, thus a 32–bit integer must be assigned to store this value. Square root computation is not a very easy task for computers compared to addition, multiplication etc. Here the operation needs a lot of computation because square root operation must be performed for every pixel of the image. For that reason, iterative square root calculation technique called Babylonian Square Root is applied. This method has been explained in section 2.5 before.

In order to justify this approximation, giving an example from the image processing software would be appropriate. Let x and y gradients be 0xF4 and 0xA5 respectively. Sum of second powers of these two numbers is 86761 and the square root of it is 295.552 precisely in decimal. Here it should be noted that another problem in Sobel operator application is that the gradient magnitude can exceed the value what an unsigned char variable can represent. That's why the calculated magnitude is later scaled down to three quarter. Iterations given below show how fast and accurate Babylonian Square Root technique is. Software always takes 180 as initial estimation because it is half of the maximum gradient magnitude.

$$x_0 \approx 180 \tag{30}$$

$$x_1 = \frac{1}{2}\left(x_0 + \frac{S}{x_0}\right) = \frac{1}{2}\left(180 + \frac{86761}{180}\right) = 1461 \tag{31}$$

$$x_2 = \frac{1}{2}\left(x_1 + \frac{S}{x_1}\right) = \frac{1}{2}\left(1461 + \frac{86761}{1461}\right) = 760 \tag{32}$$

$$x_3 = \frac{1}{2}\left(x_2 + \frac{S}{x_2}\right) = \frac{1}{2}\left(760 + \frac{86761}{760}\right) = 437 \tag{33}$$

$$x_4 = \frac{1}{2}\left(x_3 + \frac{S}{x_3}\right) = \frac{1}{2}\left(437 + \frac{86761}{437}\right) = 317 \tag{34}$$

$$x_5 = \frac{1}{2}\left(x_4 + \frac{S}{x_4}\right) = \frac{1}{2}\left(317 + \frac{86761}{317}\right) = 295 \tag{35}$$

Here very accurate result is achieved after the 5th iteration but 4th iteration is also close to the result with an error of:

$$\frac{|317-295|}{295} = 7.5\% \tag{36}$$

For the algorithm which is applied in this software, this error rate is acceptable because the error rate is proportional to how big the number is, thus it doesn't seriously affect the ratio between the gradients of different points.

Now the gradient magnitude is calculated for a given pixel, it is placed in the corresponding locations of all pixels of the output image. Gradient values are coded on the output image in grayscale format.

5.5.3. Edge detection

Edge detection is done by filtering the resultant gradient image that is obtained after the application of Sobel algorithm.

Firstly a threshold value of the filter is chosen. The gradient values which are below this threshold will be coded as black, and the values that are greater than the threshold can either be coded as pure white or the gradient value itself. Gradient image is in fact the edge detected image with a zero threshold. Setting the threshold value to maximum level will of course result in a pure black image.

Applying the Sobel operation over the image twice results in an approximation of Laplace transform of the image, but in this work it is not performed.

6. EXPERIMENTS AND RESULTS

In this section the acquired images, their transformations to monochrome format, gradient detected format, and edge detected formats with different parameters are represented.

6.1. Captured image

A captured image of a paper that consists of the primary colours red, green, and blue on it is depicted below in Figure 22.

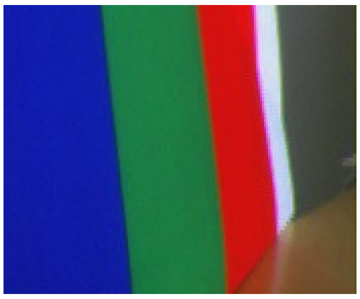


Figure 22. A captured image depicting the primary colours of RGB colour space.

This image is in a raw form after receiving from the camera. It was sent directly to the computer as a bmp file in 176 x 144 resolution.

6.2. Monochrome transformed image

As it is described in the software chapter, the monochrome transformation that the software computes is obtained by using the multiples of two in order to achieve high performance. Since the results are obtainable by implementing bitwise operations, using fractional numbers and using mathematical operators are avoided.

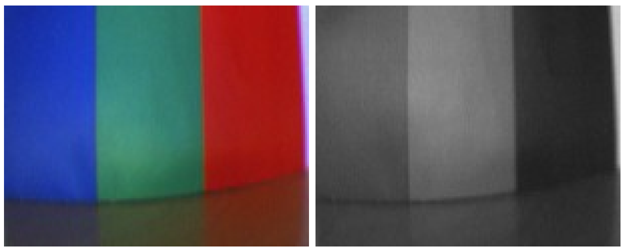


Figure 23. RGB and monochrome coded images of the same object.

The Figure 23 above shows two pictures of the same object taken from the same location at different times. The image on the left is captured and directly sent to the computer as a bmp file without any operation, and the one on the right is captured and transformed to monochrome format before being sent to the computer as a bmp file. The transformation is done by using monochrome.

Another example of the transformation is shown in Figure 24. Notice that the output image from the vision sensor is not scaled to match with the horizontal versus vertical ratio of the object. The pen seems thicker than it really is. This situation can be corrected by finding the camera calibration parameters and scaling the image with those parameters.

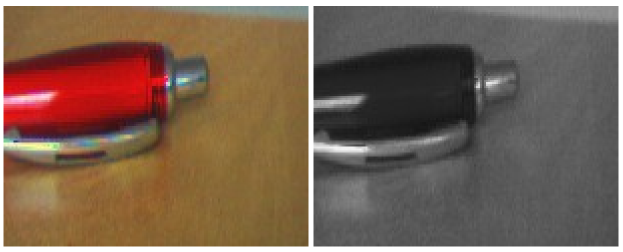


Figure 24. Two monochrome transformed images. Camera doesn't provide scaled images in 8–bit RGB mode.

Though the images were transformed by avoiding more complicated arithmetic, results seem satisfactory.

6.3. Gradient calculated image

As described in section 2.4, gradient of the image is calculated with horizontal and vertical convolution kernels. Using a 3x3 kernel over the monochrome images, first derivatives of the images are approximated as shown illustrations in this section.

The gradient magnitudes of the images below are computed by using Babylonian Square Root approximation with four step iteration. This method was already described in 2.5. It allows easy square root calculation without dealing with a lot of computations in order to find precise square root values of big numbers. Since this operation is done for every single pixel, it is quite beneficial for performance.

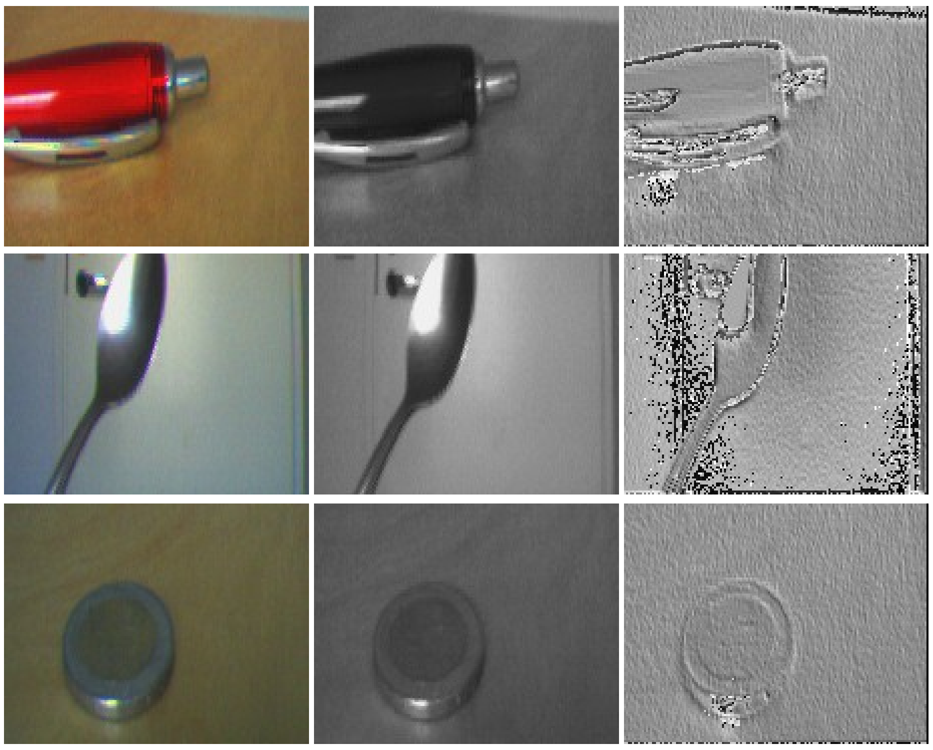


Figure 25. Original, monochrome, and the first derivative (gradient) of some images.

Here it should be noted that after the Sobel operation, the last column and the last raw of the gradient images are missing. In order to compute a gradient value for those pixels, there should be further pixels to calculate how much they differ from each other. That's why those values can not be computed.

6.4. Edge detected image

Edge detection of an image is done by applying a filter over the gradient calculated image. Application of the filter is as simple as coding black level for the pixels that have a gradient value lower than a threshold level.

Different threshold levels introduce various results with different noise levels. Depending on the application and the environmental conditions, the optimum threshold level should be determined.

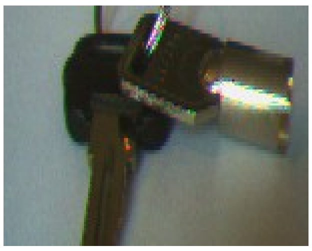


Figure 26. Original image in RGB format.

The results below illustrate the edge detection for the object in Figure 26 with different threshold levels.

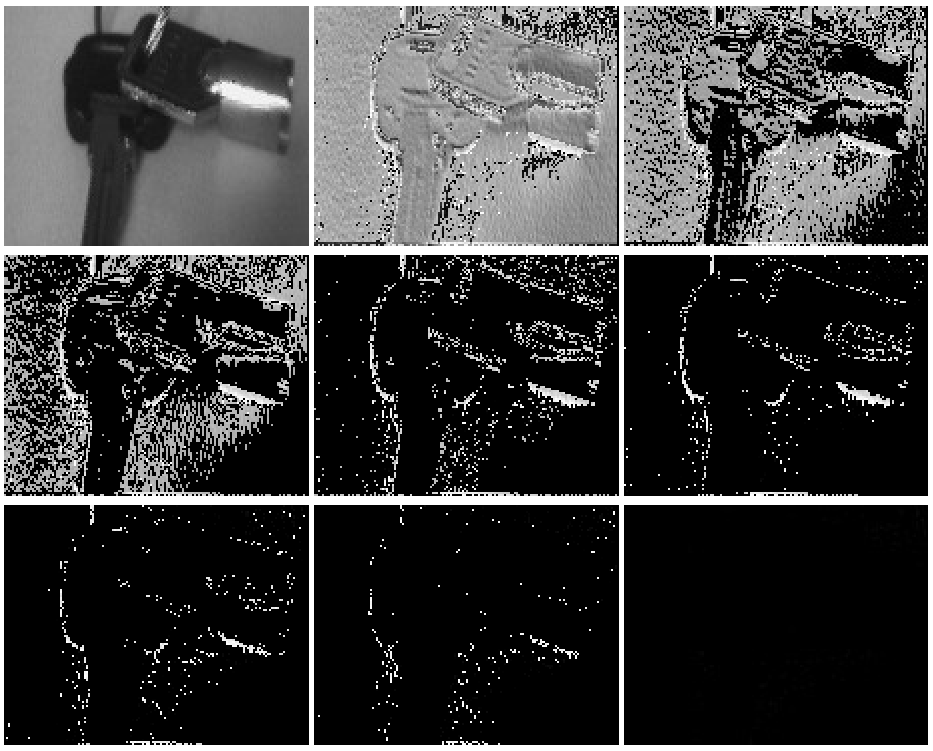


Figure 27. Original, monochrome, and edge detected images using various parameters.

The gradient image here is just a special case of the edge detected image with zero filtering level. Parameters of the edge detected images from top left to bottom right are 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, and lastly 0xFF in hexadecimal. Filtering the image with maximum threshold level results in black image.

6.5. Data compression on edge detected images

Data compression in wireless sensor networks is directly related to the transmission amount that's why it is quite important in the means of power consumption. Compressing the amount of transmitted data doesn't only mean the reduction of transmission power, but also means increased security. In Figure 27 the edge detected pictures either have white or black colours. Instead of transmitting the whole picture, just the white pixel coordinates can be transmitted.

For any pixel, both vertical and horizontal coordinate can be represented by one byte, totalling in two bytes. But in a grayscale image, one pixel value is coded with a single byte, for that reason it may not always be beneficial to transmit the coordinates instead of the whole image.

Assume that the pictures shown in Figure 27 starting from the top left, down to bottom right are numbered from 1 to 9. Figure 28 shows the amount of transmission data in both the full transmission case and the coordinate transmission case for each picture.

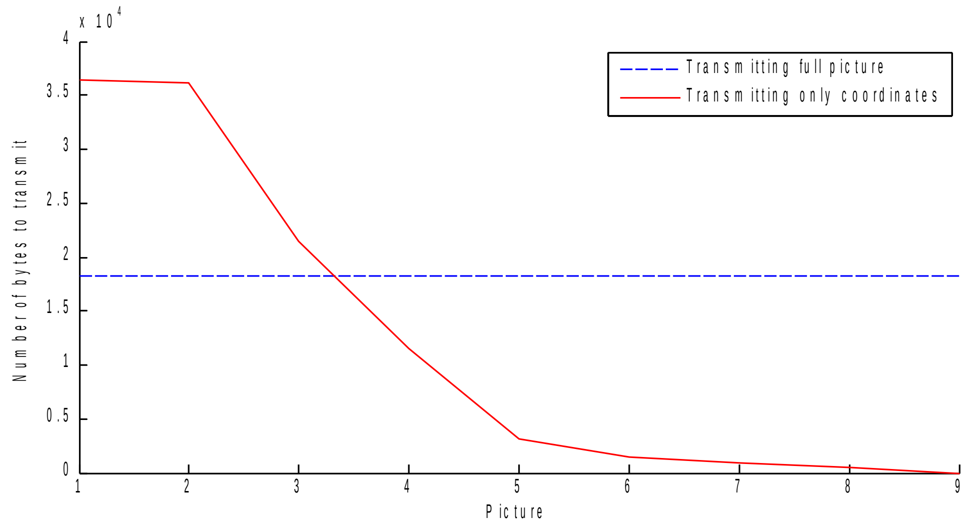


Figure 28. The amount of data to transmit for pictures 1–9.

Since the first three pictures have more white pixels than the black pixels, it is not beneficial to choose coordinate transmission. In that case, transmitting only the black pixel coordinates would inverse the situation. Since this work doesn't focus deeply on transmission algorithms, only the idea is presented.

The ratios between full transmission and coordinate transmission for each image are listed in Table 9 by percentages.

Table 9. Image sizes using coordinate transmission compared to original from.

Picture	Image size using coordinate transmission
1	200%
2	198%
3	117%
4	63%
5	17%
6	8%
7	5%
8	3%
9	0%

Although using the threshold levels in number 8 and 9 may seem the best option regarding to image sizes, they probably may not represent sufficient information. The optimum edge detection threshold must be chosen depending on the application criterias.

6.6. Usage of EMC and SRAM

The current configuration of the design that generates the images represented in this work, the EMC of the processor and SRAM weren't used. In order to demonstrate that the data transfer between the processor and the SRAM is possible, a test is represented in this section.

The code used in the test is given below:

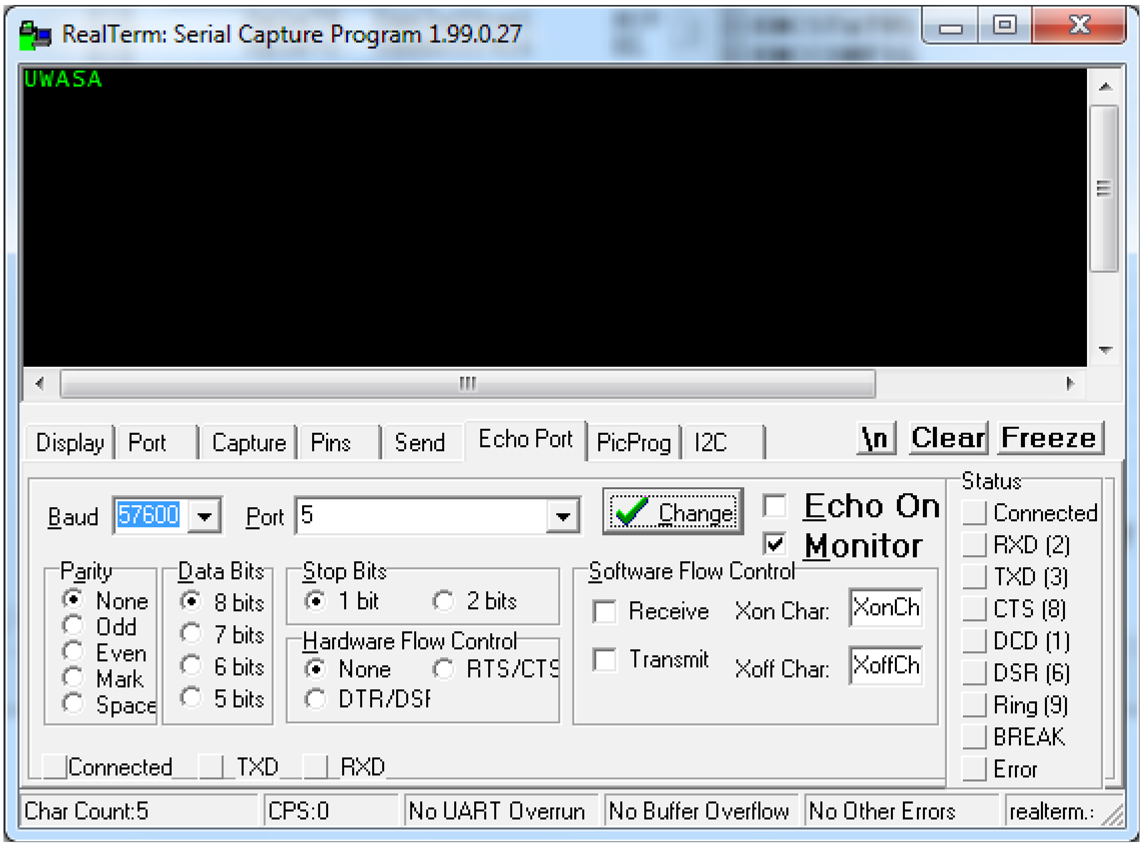
```
int c,g;

frame[0][0] = 'U';
frame[0][1] = 'W';
frame[0][2] = 'A';
frame[1][0] = 'S';
frame[1][1] = 'A';

s_p = &space0[0];
switch_track(emc_to_sram);
for(c=0; c<2; c++)
{
  for(g=0; g<3; g++)
  {
    *s_p = frame [c][g]; s_p++;
  }
}

switch_track(sram_to_emc);
UART1Init(57600);
u_p = &space0[0];
UARTSend((unsigned int)(5));
```

Here the software creates a small two–dimensional array called *frame* on the internal memory, and assigns the characters of *UWASA* word to its elements. Then the SRAM pointer *s_p* is assigned to the previously allocated external memory block on the SRAM. The function called *switch_track* here is necessary for data direction configuration on the test board. After that, using EMC, the elements of the *frame* matrix are written from internal memory to the external SRAM memory. Following that operation, the UART interface is initialised, and the UART pointer *u_p* points to the external memory that is previously written by *s_p*. Finally, the processor sends those values to computer by directly reading from external SRAM in order to ensure that the operation is successful.



```
UWASA
```

Baud 57600   Port 5

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR

Software Flow Control: Receive, Transmit; Xon Char: XonCh; Xoff Char: XoffCh

[✓ Monitor]

Connected TXD RXD

Char Count:5 | CPS:0 | No UART Overrun | No Buffer Overflow | No Other Errors | realterm.:

Figure 29. Characters of UWASA were sent directly from external SRAM to computer.

In this test a memory bank on the SRAM was allocated as a variable called *space0*. This memory location behaves like it internally exists inside the processor. As illustrated in here, a variable on that memory location can be created and used. The EMC peripheral of the processor makes this operation possible.

7. CONCLUSIONS

It would be appropriate to say that this work has achieved two major goals. The primary goal at first sight was to design a slave module for the UWASA Node in order to make it capable of image acquisition, transmission, and processing.

The primary goal has been achieved when the proposed hardware architecture which consists of a camera board, a processor and a new designed hardware interface has been successfully produced as a prototype and tested. The test board has provided a working platform to identify and troubleshoot the problems which had been faced during the development process. The test board and the camera board together have represented the behaviour of the slave camera module which may be adapted to UWASA Node. If a camera module needs to be produced based on the work here, it is verified that the schematic structure can be applied without any problem. Even though the test board was produced in rather big dimensions to enable easy production, the slave module can be designed in a form as small as approximately 15 cm^2.

The secondary goal attained has been the development of convolution, monochrome image transformation, gradient calculation, and edge detection algorithms. Instead of doing complex calculations in a formalised way, some easy computation methods for those algorithms have been introduced. Here it should be noted that those easy computation methods were chosen and employed appropriately to prevent mistaken results. In the end, all those methods verified to fulfil the requirements.

Although the images provided by the vision sensor don't have very good quality due to the fact that it was designed more than ten years ago, the feasibility of the image processing inside the UWASA wireless sensor node has been proven.

In future, the edge detection method demonstrated in here can be developed further into contour detection, then into image segmentation, and finally into object identification. On the contrary, it would also be a good decision to optimise and improve the capabilities of the existing algorithms before moving further. Although the hardware can currently be produced, the efficient utilisation of the SRAM chip, and redesigning the test board with minor improvements can bring this design into a finalised work. In case the camera board has to be replaced, moderate level changes would be necessary. In that case, of course a new prototype should be tested.

REFERENCES

Bradski, Gary & Kaehler, Adrian (2008). *Learning OpenCV*. 1st Ed. California etc.: O'Reilly Media Inc.

CMUcam3. CMUcam3: Open Source Programmable Embedded Color Vision Platform [online] [cited 25 April 2011]. Available from the Internet: <URL: http://cmucam.org/chrome/site/images/gallery_imgs/front.jpg>.

Çuhac, Caner, Hüseyin, Yiğitler, Reino, Virrankoski & Mohammed, Elmusrati (2010). Camera Integration to Wireless Sensor Node. In: Aalto University Workshop on Wireless Sensor Systems [online]. Aalto University Wireless Systems Group. Helsinki: Aalto University.

Culuricello, Eugieno & Andreas G. Andreou. (2006). CMOS image sensors for sensor networks. *Analog Integrated Circuits and Signal Processing*. 49:1. 39–51.

Fowler, David & Eleanor, Robson (1998). Square Root Approximations in Old Babylonian Mathematics: YBC 7289 in Context. *Historia Mathematica* 25, 366–378.

Frontier. bmp graphics files explained. [online] [cited 18 March 2011]. Available from the Internet: <URL: http://www.frontiernet.net/~fys/bmpfile.htm>

Kanopoulos, Nick, Nagesh, Vasanthavada & Robert L. Baker (1988). Design of an Image Edge Detection Filter Using the Sobel Operator. *IEEE Journal of Solid–State Circuits* 23:2, 358–367.

Maxim. Maxim Full Datasheet. [online] [cited 23 April 2011]. Available from the Internet: <URL: http://datasheets.maxim-ic.com/en/ds/MAX1722-MAX1724.pdf>.

Olimex. LPC2378STK [online] [cited 25 April 2011]. Available from the Internet:

<URL: http://www.olimex.com/dev/index.html>.

Serway, Raymond A. & John W. Jewett (2010). *Physics for Scientists and Engineers with Moders Physics*. 8th Ed. California. Cengage Learning.

Sheldon, M. Ross (2010). *Introduction to Probability Models.* 10th Ed. California etc.: Elsevier Inc.

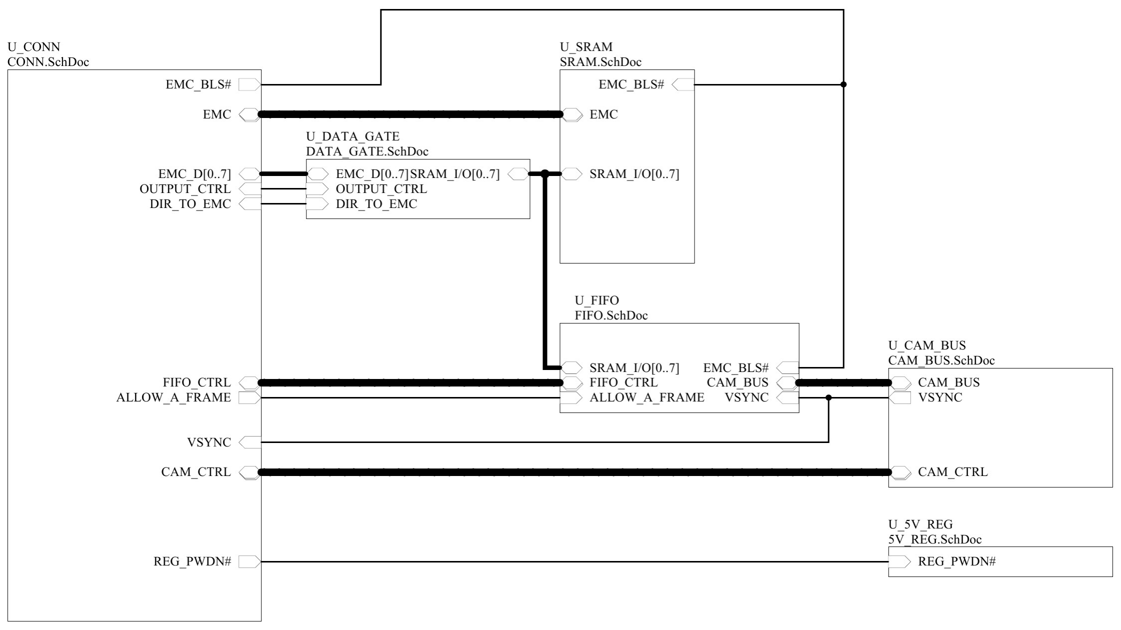
Wikipedia. Bayer filter [online] [cited 18 April 2011]. Available from the Internet: <URL: http://en.wikipedia.org/wiki/Bayer_filter>.

Yiğitler, Hüseyin. (2010a). *The UWASA Node Reference Manual* [online]. 1st Ed. Vaasa–Finland: University of Vaasa, 2011. Available from the Internet: <URL: http://wsn.tkk.fi/en/software/uwasa_manual.pdf>.

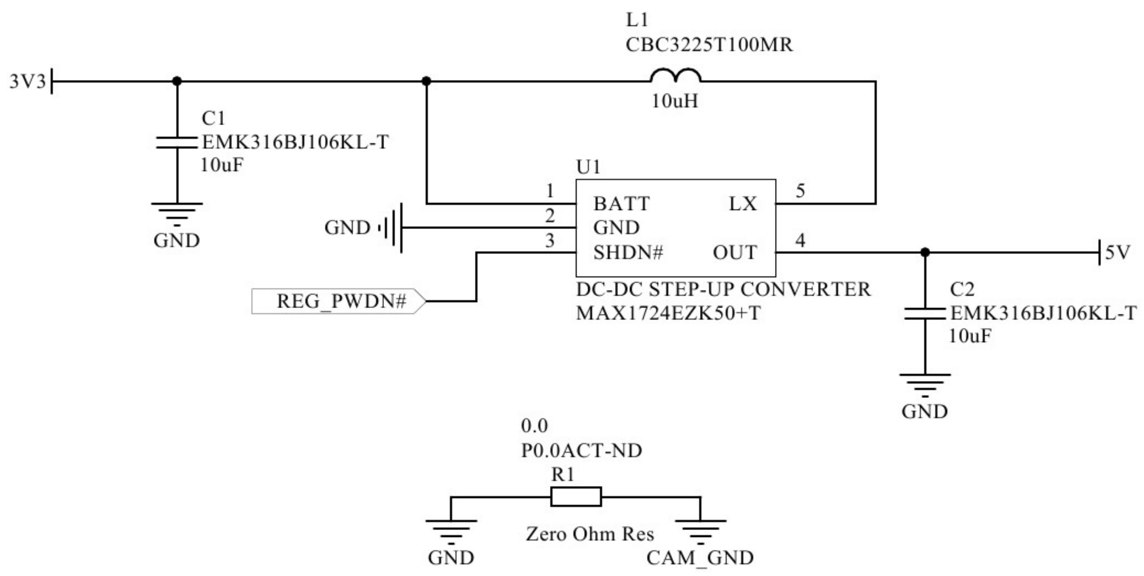
Yiğitler, Hüseyin, Reino, Virrankoski & Mohammed, Elmusrati (2010b). *Stackable Wireless Sensor and Actuator Network Platform for Wireless Automation: The UWASA Node*. In: Aalto University Workshop on Wireless Sensor Systems [online]. Aalto University Wireless Systems Group. Helsinki: Aalto University.

APPENDICES

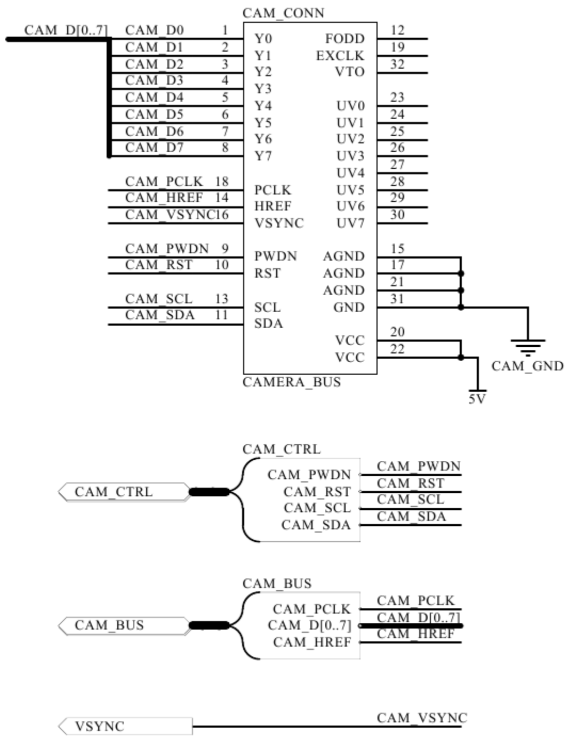
APPENDIX 1. Schematic design of the hardware blocks on the board.



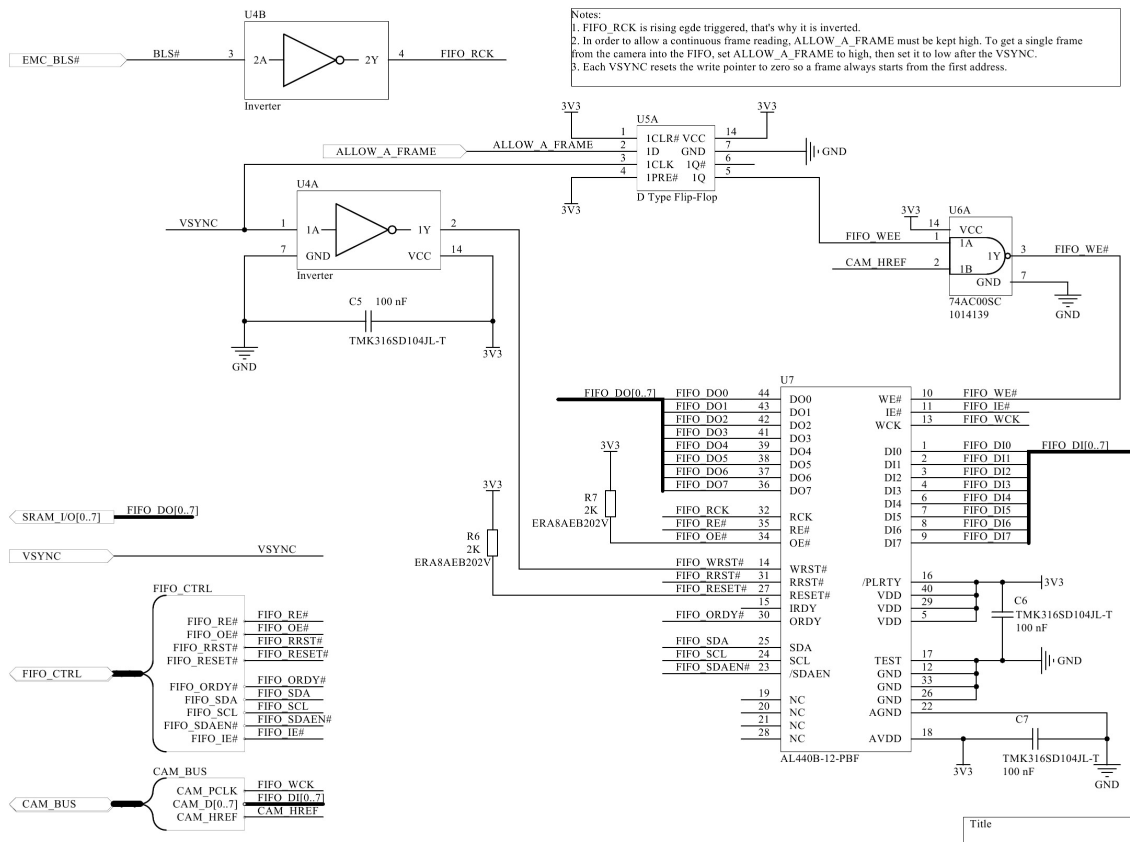
APPENDIX 2. Schematic design of the 5 V regulator.



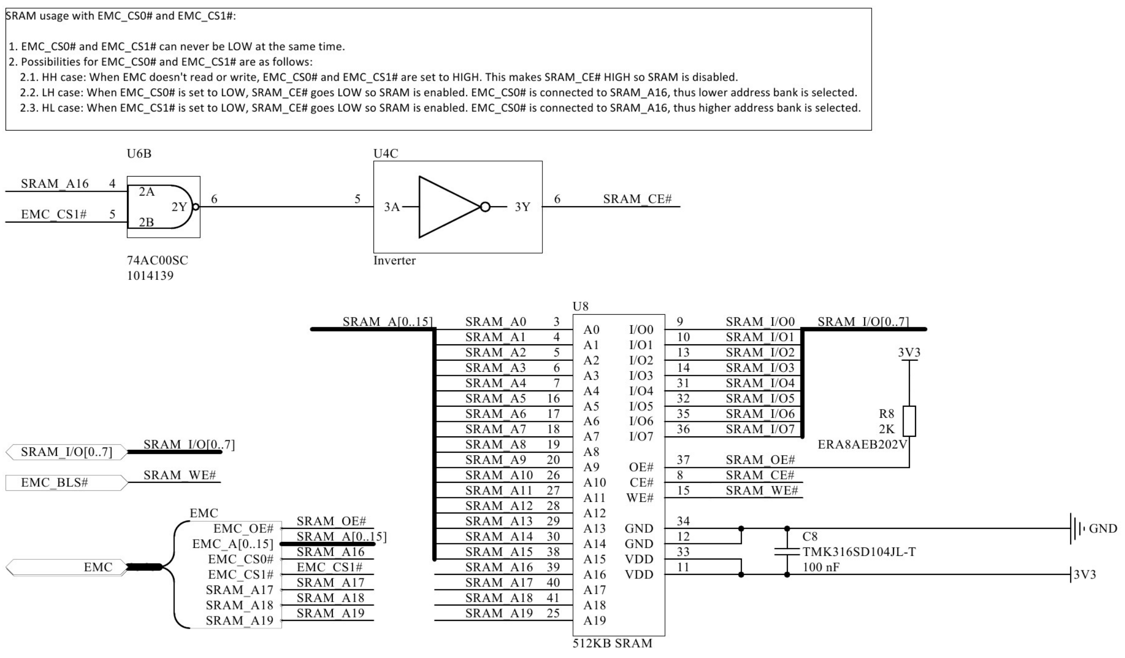
APPENDIX 3. Schematic design of the camera bus.



APPENDIX 4. Schematic design of the FIFO.


Notes:
1. FIFO_RCK is rising egde triggered, that's why it is inverted.
2. In order to allow a continuous frame reading, ALLOW_A_FRAME must be kept high. To get a single frame from the camera into the FIFO, set ALLOW_A_FRAME to high, then set it to low after the VSYNC.
3. Each VSYNC resets the write pointer to zero so a frame always starts from the first address.

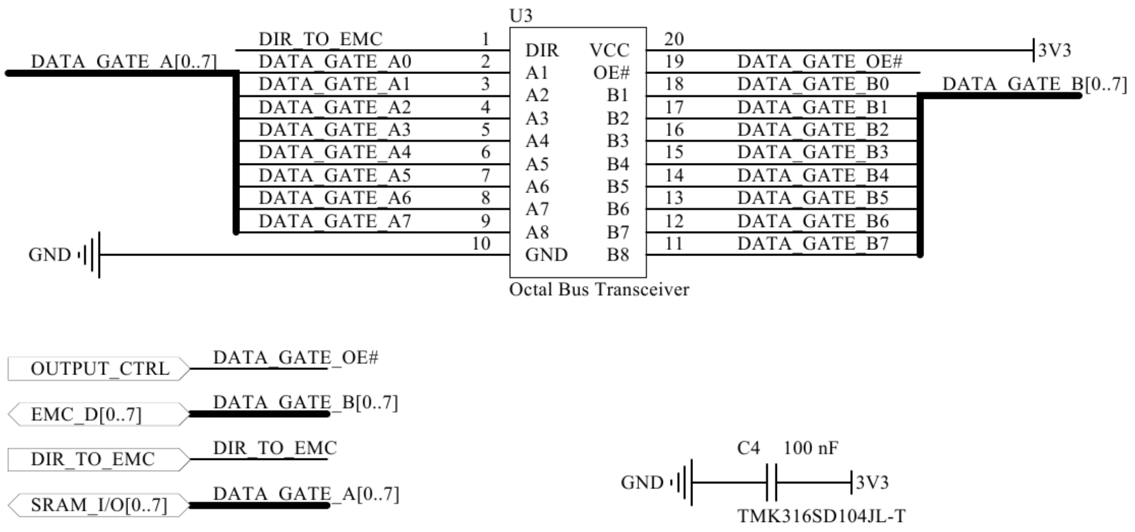
APPENDIX 5. Schematic design of the SRAM.



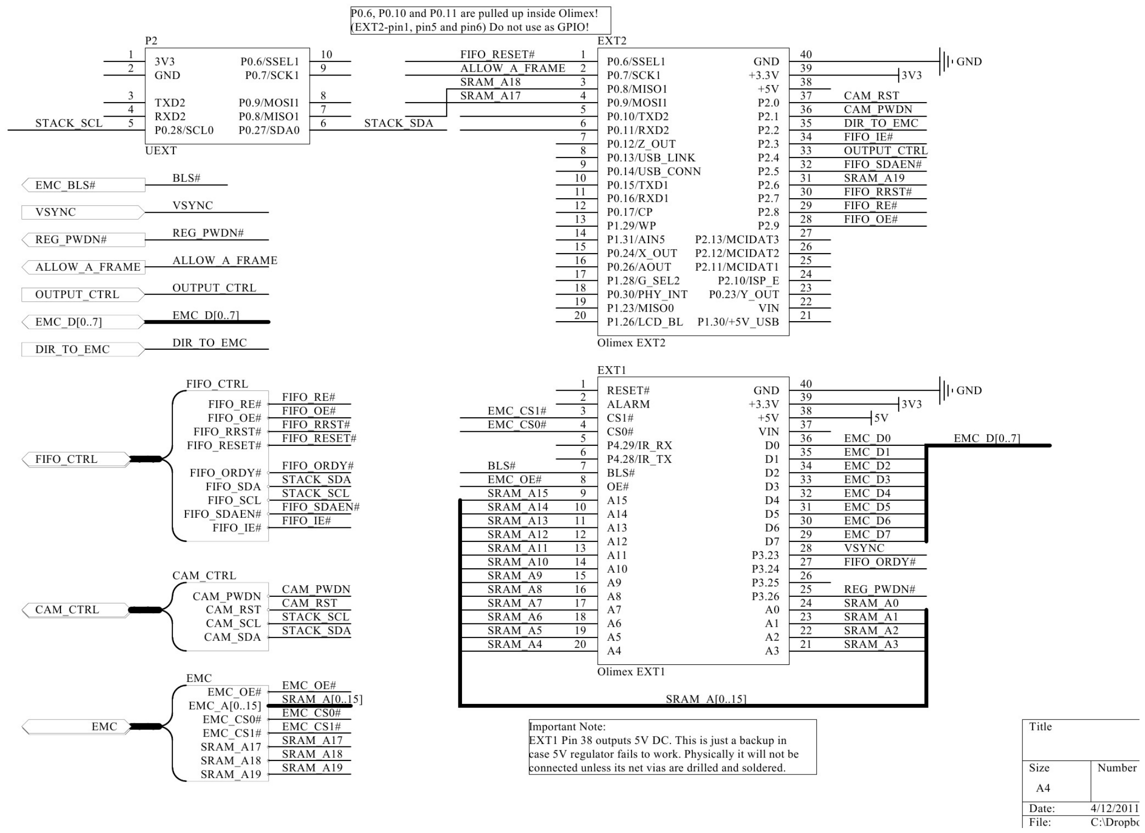
SRAM usage with EMC_CS0# and EMC_CS1#:

1. EMC_CS0# and EMC_CS1# can never be LOW at the same time.
2. Possibilities for EMC_CS0# and EMC_CS1# are as follows:
   2.1. HH case: When EMC doesn't read or write, EMC_CS0# and EMC_CS1# are set to HIGH. This makes SRAM_CE# HIGH so SRAM is disabled.
   2.2. LH case: When EMC_CS0# is set to LOW, SRAM_CE# goes LOW so SRAM is enabled. EMC_CS0# is connected to SRAM_A16, thus lower address bank is selected.
   2.3. HL case: When EMC_CS1# is set to LOW, SRAM_CE# goes LOW so SRAM is enabled. EMC_CS0# is connected to SRAM_A16, thus higher address bank is selected.

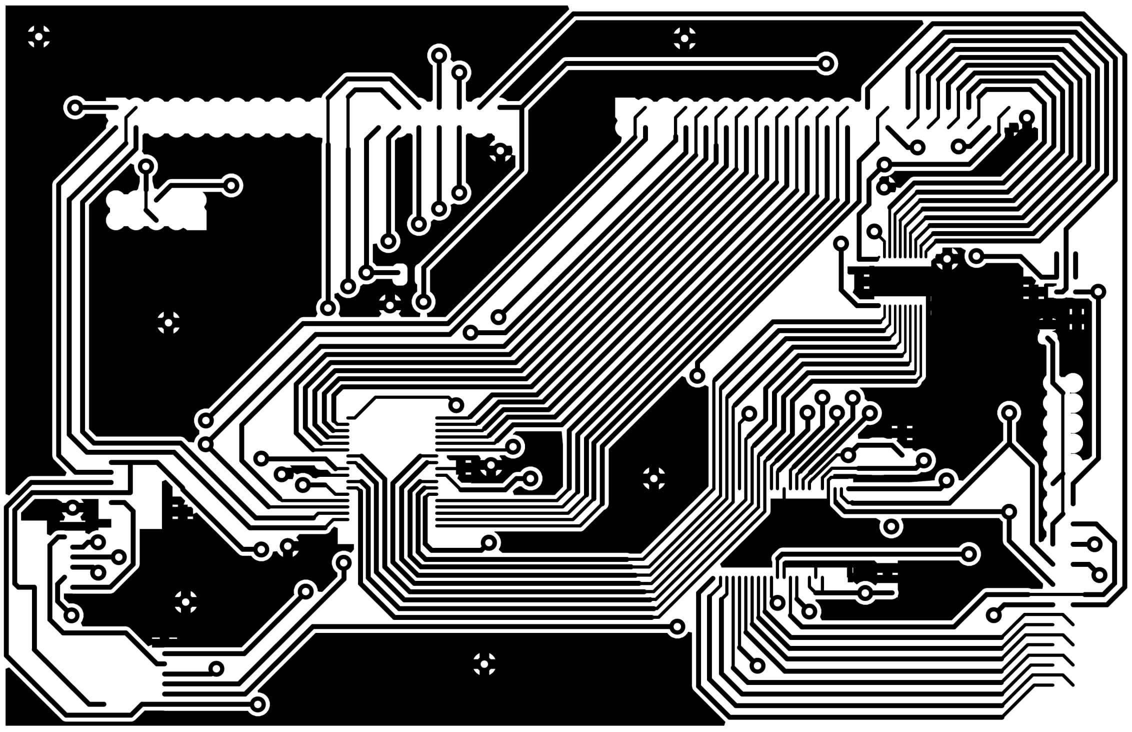
APPENDIX 6. Schematic design of the switching circuit.



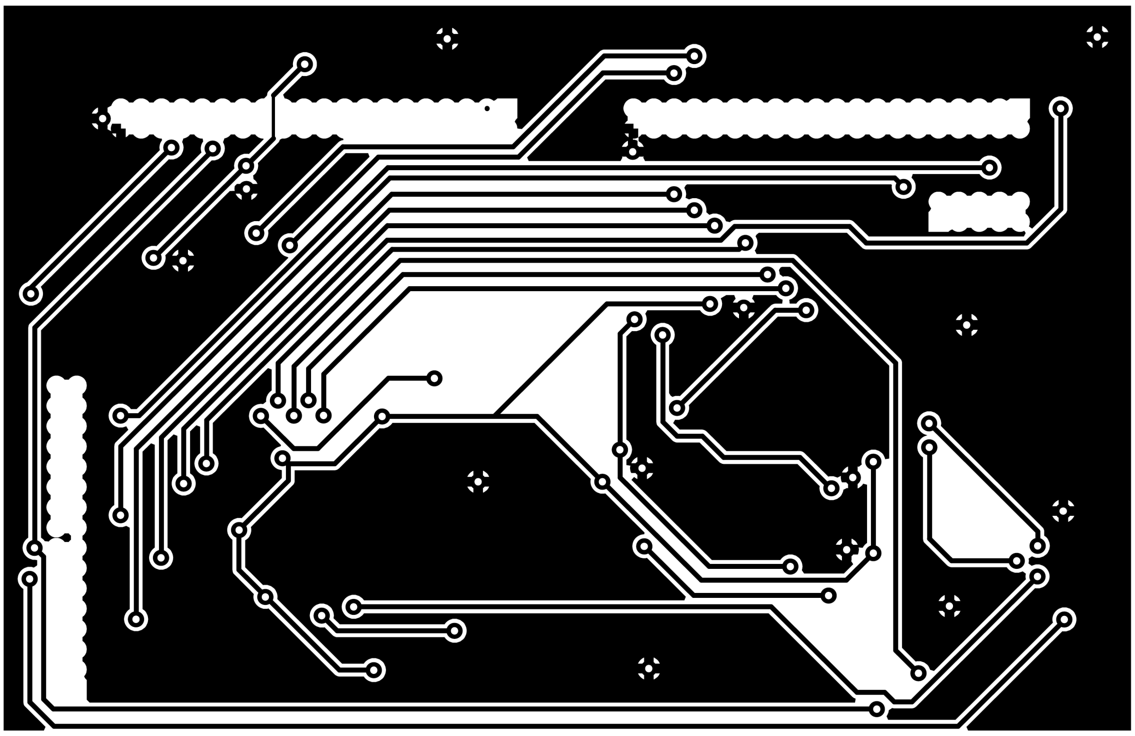
APPENDIX 7. Schematic design of external pin processor connections.



APPENDIX 8. PCB top layer.



APPENDIX 9. PCB bottom layer.



APPENDIX 10. Top and bottom views of the test board.