

Intuitively Controlled Robotic Arm

Stefan Wojciechowski(403546963), Jason Tan(003670742),
Thomas Markes(103563082)
Computer Science - UCLA

March 2012

Contents

1	Project Background	1
2	Specification	1
3	Overview Description	1
4	Design Approach and Challenges	2
5	Mathematics	8
6	Conclusion	9
7	Algorithms	9

List of Figures

1	Shoulder Rotation	3
2	Shoulder Movement	3
3	Elbow Movement	4
4	Elbow Conditions	4
5	Clamp Movement	5
6	ICRA	6
7	Angle Calculation	7
8	Elbow Movement	12
9	Shoulder Y	12
10	Shoulder X	13
11	Module and Slice Count	14
12	FPGA Modules	15
13	FPGA Modules Cont.	16

Abstract

Robotics have become an increasingly prevalent technology in the current decade. Automotive, medical, and heavy industries currently use robotics to facilitate workers in the completion of their tasks. Most of these robotic appendages utilize non-standardized control interfaces with a myriad of joysticks, switches and buttons to operate a robotic arm. This is not intuitive and requires training for an individual to properly use. Furthermore with the increasing complexity of these robotic interfaces safety of operators and those in proximity of these robotics must be considered. We propose an intuitively controlled robotic arm (ICRA) that maintains an intuitive interface for the control of robotic appendages through the use of real time hardware image processing of a human arm that is then mimicked by the robotic arm.

1 Project Background

Human tracking robotic interfaces are currently a heavily studied segment of robotics with several universities conducting research in this field. Many have recognized the importance of robotics and current uses. Robotics are currently used in medical applications for surgery, and have seen action on the battlefield in both Iraq and Afghanistan. Various heavy industries make use of robotic arms as well, namely the automotive industry in their manufacturing plants as well as countless other industries. The use of robotics is almost universal. Despite the prevalence, most of these robots are not controlled by an intuitive means. They require a layer of abstraction in the use of joysticks and switches to control what would otherwise be a natural action. We seek to explore this possibility.

With the advent of inexpensive technologies such as the Microsoft Kinect, human body tracking has become a widely available technology. However many of these projects have a noticeable delay in reaction time and require the operator to move fairly slowly for proper tracking to occur. The use of dedicated hardware through an FPGA would allow much faster image processing and therefore a minimized delay in reaction of the robotic components. This is ideal for intuitive control as a delay in response dampens the intuitivity of the control.

2 Specification

Using one FPGA, one camera, and a robotic arm we planned to implement limited 3-D axis of motion in real time. The robotic appendage would be able to raise and lower an arm on the shoulder joint (figure2), simultaneously bend an elbow naturally in spite of shoulder position (figure4), as well as rotate the shoulder joint on the z-axis (figure1), as if to swing an arm. A clamp appendage would be added to simulate a hand (figure5), allowing ICRA to grasp objects. The goal was for real-time response from the robotic arm, and intuitive, precise tracking so that little to no training would be necessary for the use of the robotic arm.

3 Overview Description

ICRA utilizes a Xilinx Vertex-II Pro Development System. In combination with micro-servos, video camera, and monitor, they provide the complete hardware package for ICRA to function. All coding was implemented in VHDL. The goal of ICRA was a simple humanoid robotic arm to follow literal human movements and perform the same movement in parallel. This experiment was designed to be a proof of concept behind the possibilities of intuitive control that could facilitate various industries that may benefit from less training and an increase in

safety that would be associated with intuitive control.

4 Design Approach and Challenges

The image processing design in ICRA required the tracking of multiple points located on the human arm to accurately determine a human arm position. It was decided that pixel color tracking would be the best option, with each arm joint marked with a specific color that could be chosen for the image processing module to track. Each color is submitted to a calibration mode, which determines which color is used to track which joint. The main challenge of the image tracking was threshold determination (algorithm 8). The testing environment was less than ideal with many objects of color in full view of the camera. On occasion the camera would pick these objects to track as opposed to the arm markers. A better algorithm that takes into account proximity of pixels may eradicate many of these issues.

ICRA's motion is enabled via servos. These servos are directly connected to the FPGA. Known drivers that were available online did not work correctly with the FPGA so a custom driver was implemented to drive the servos. Due to time and financial constraints the servos chosen for the project were of dubious quality and did not include a specification sheet. These servos used timed impulses to determine degrees 0-180. Significant use of the oscilloscope was necessary to approximate the impulse range for the servo to operate properly. Once this was determined, we were able to drive our servos via the FPGA.

It then followed that with the completion of the servo drivers and image tracking, we combined the two. Simple movements, such as raising and lowering an arm were explored. It was soon realized the importance of position and scaling to achieve accurate results. It was determined that the distance from the camera should remain fixed and that certain issues, such as out of bounds conditions, would have to be ignored for the sake of time.

With servos moving with the help of image tracking, A durable robotic arm was constructed. Using form board for limbs, we were able to construct a very light, rigid, and relatively strong robotic arm to reflect our ultimate goal. The robotic arm matched specifications with a shoulder joint, shoulder z-axis rotation, elbow joint, and clamp.

ICRA's shoulder has two axis of motion. It was designed to raise and lower as a human shoulder would, as well as rotate, as if to pass an arm across the body. Shoulder raising and lowering was quickly determined to be a matter of scaling the difference between the shoulder Y position and elbow Y position with relation to being raised or lowered passed it's 90 degree baseline (See algorithm 1). The rotation of the shoulder joint however, was not as straight forward. The rotation was determined by the difference between the elbow X position

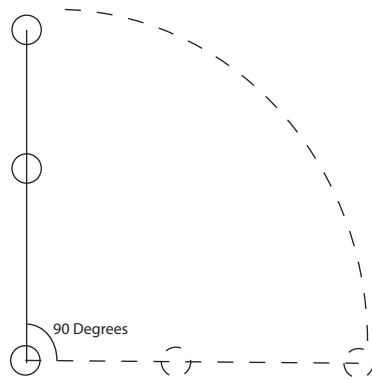


Figure 1: Shoulder Rotation

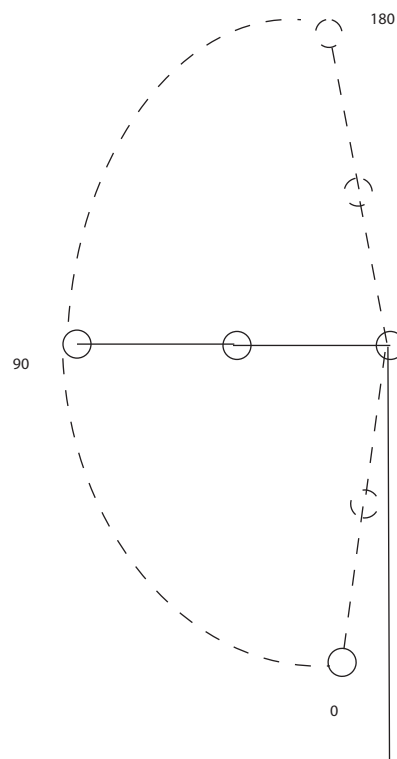


Figure 2: Shoulder Movement

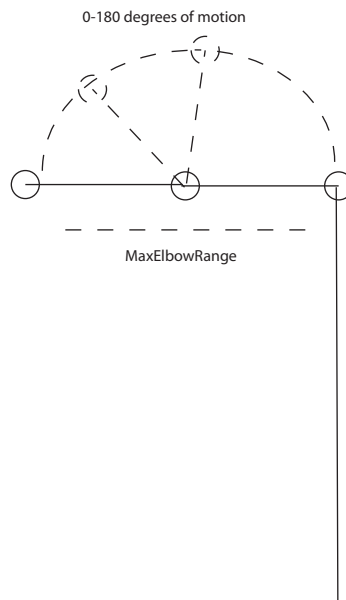


Figure 3: Elbow Movement

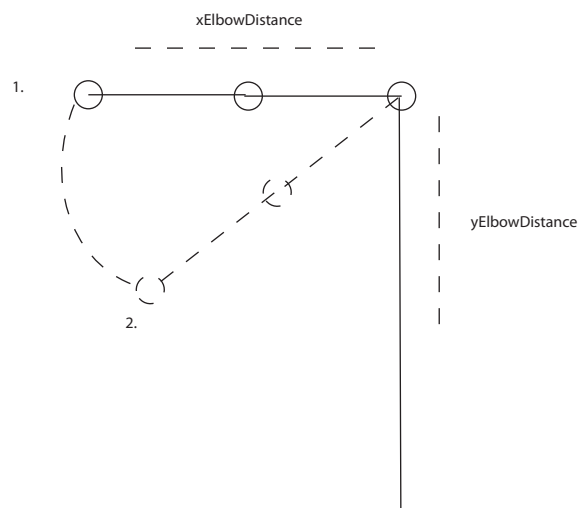


Figure 4: Elbow Conditions

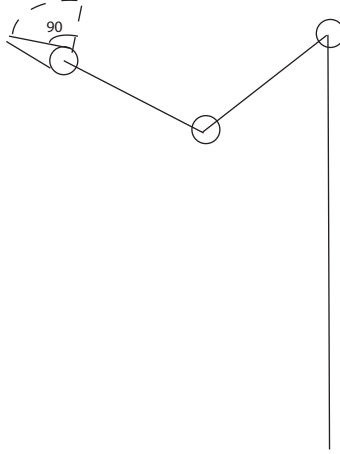


Figure 5: Clamp Movement

and shoulder X position (See Algorithm 1). As the difference grew smaller, one could determine a proper scaling to achieve 0-90 degree rotation. This alone however would not suffice. A penalty was implemented because the same values used to determine shoulder Y-axis movement would also effect shoulder rotation. This penalty ensured relative stability of the shoulder Rotation despite shoulder Y-axis movement.

ICRA's elbow joint was then implemented simultaneously on another FPGA to increase productivity. The initial design utilized the wrist X coordinate with respect to the shoulder X coordinate to determine a bend or extension of the elbow. This worked well for the base case of a straight forward arm (figure 3). However, this was not enough when the change in position of the shoulder was included (figure 4). The straight forward approach was to use the actual distance between the wrist and shoulder positions to determine an accurate value by which to scale the bend. This proved impossible to implement because of FPGA timing issues. We considered changing the sampling to allow for a more accurate and straight forward calculation to be made, but instead opted for rough approximations.

It was determined that a comparison of the x coordinate difference vs the y coordinate distance could give a basic approximation of the elbow bend. By determining the relative lengths of each difference, we determined which axis carried more weight in determining the bend of the elbow (figure 4). This proved

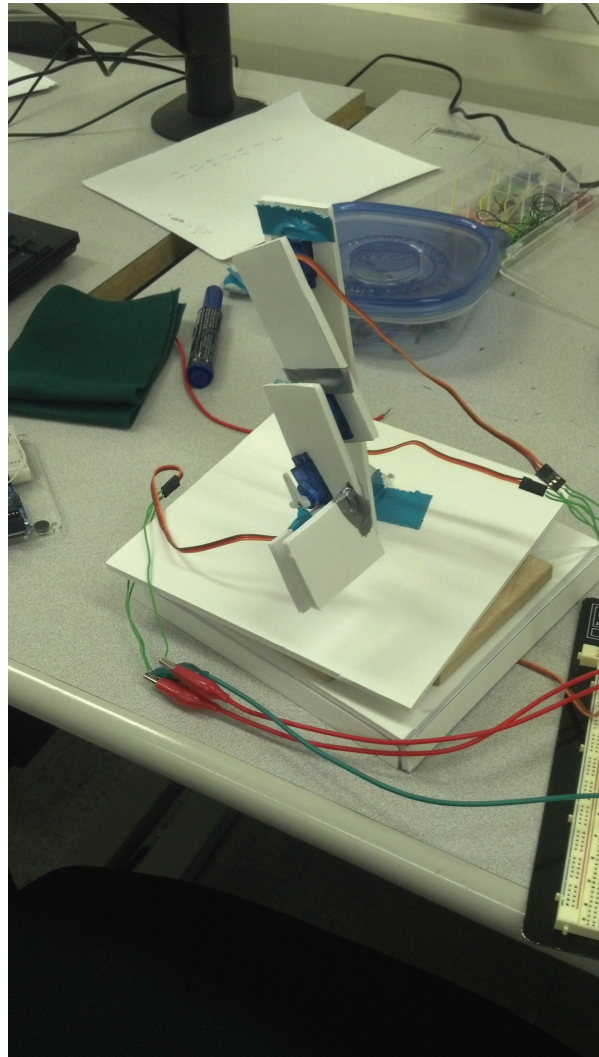


Figure 6: ICRA



Figure 7: Angle Calculation

to work very well in conjunction with the shoulder joint. However, with the addition of the shoulder rotation, a penalty had to be added to ensure that an accurate result would be returned (Algorithm 3).

The clamp was designed to be closed at 0 and open at 90 degrees (Figure 5). Clamp movement was implemented by detecting the appearance of a color near the wrist. By letting a hand reveal a particular color near the wrist marker, we were able to implement a boolean expression for the opening and closing of the clamp. If the color appeared, open the clamp, otherwise close the clamp (algorithm 5).

The last design aspect that required implementation was the special case of a completely forward facing arm. In this case, many of the markers used to track arm motion would be lost given this orientation. The initial approach was to include another marker on the front of the hand and using pixel detection to determine depth. This however, was not enough to interpret the full degrees of motion that the arm could perform. This was considered too difficult of a condition to address and is still left open for improvement.

5 Mathematics

The complex mathematics that we were originally going to use took advantage of complex trigonometry and the Law of Cosines to calculate the three important angles for our servos. This algorithm would receive both an X and Y coordinate of three of our markers (Hand, Elbow, Shoulder) and output angles for the elbow, shoulder-Y, and shoulder-X as shown in figures 8,9 and 10. The Law of Cosines can return any angle of any given triangle as long you have the lengths of each triangle side. The angle that it returns is the one whose opposing triangle side is given as c in the equation.

For the Elbow Angle triangle, the three sides are the length from the hand to the elbow (labelled as a), the length from the elbow to the shoulder (labelled as b), and the length from the hand to the shoulder (labelled as c). For the Shoulder-Y triangle, the three sides are the length from the elbow to the shoulder (labelled as a), the Y-distance between the shoulder and the elbow (labelled as b), and the X-distance between the shoulder and the elbow (labelled as c). For the Shoulder-X triangle, the three sides are the length from the elbow to the shoulder (labelled as a), the X-distance between the shoulder and the elbow (labelled as b), and the Y-distance between the shoulder and the elbow (labelled as c). In order to make sure that our mathematics would work, we took pictures of expected poses for the arm in the POV the camera would see. We did the work in Google Spreadsheet and used Paint to estimate the X and Y values that our image processing algorithm would give us based on markers we drew into the pictures. Our complete results are available upon request. We soon found that VHDL did not have any synthesizable trigonometric functions and we had

already decided that we would not have the time for the passing of information between VHDL and C. Thus, we decided to use a lookup table (LUT) to generate a unique angle for all possible right-side evaluations for the Law of Cosines. In this case, we would get a value for the right side given our coordinate data from the image processing and use it to generate angles for the arm. Our first step in completing this task was writing up solvable unique values for each angle (see For acos LUT spreadsheet). Calculations revealed that the cosine of every integer angle between 0 and 180 degrees can be uniquely represented by an integer if the cosine value is multiplied by 10,000. Furthermore, by adding 10,000 to that new value, we could have each value be a positive integer. This helps by letting us use unsigned vectors without worry. Using this table, for example, would let us input 19,511 and receive the angle '18 in degrees. Because of the way the servos worked, we could only allow angle inputs in integer form. To accommodate for cosine values that would normally have a decimal angle in between two integers, we would use a function to round the input to the closest allowable LUT input. We had additionally written synthesizable divide and square root functions in VHDL for use in calculating some of the lengths needed for our calculations. However, this is where we ran into timing issues with Xilinx and were forced to decide between more processing speed to allow our complex mathematics or scrap our complex mathematics to maintain our real-time image processing. Because our main project goal was an intuitive UI design for the arm, we prioritized the real-time calculations and thus abandoned the complex mathematics strategy.

6 Conclusion

Through subsequent testing and use, it was determined that intuitive control of a robotic appendage was possible. While hardware constraints forced the use of approximations for movement, relative precision was maintained. ICRA was ultimately able to rotate approximately 90 degrees as well as raise, lower, extend, and grip objects with the human interface.

7 Algorithms

$$\cos(\theta) = \frac{a^2 + b^2 - c^2}{2ab}$$

Appendix

Video of ICRA is available here <http://www.youtube.com/watch?v=4rPAXzmuXr4&feature=youtu.be>.

Algorithm 1 Shoulder Y Axis Movement

```
if shoulder and elbow markers have been found then
  if ShoulderY > ElbowY then
    ShoulderPos = 90 - 90 * abs(elbowY - shoulderY)/MaxElbowRange
  else
    ShoulderPos = 90 + 90 * abs(elbowY - shoulderY)/MaxElbowRange
```

Algorithm 2 Shoulder Rotate Movement

```
if shoulder and elbow markers have been found then
  rotatePos = 180*abs(elbowX-shoulderX)/348 + abs(90-shoulderPosition)/2;
```

Algorithm 3 Elbow Movement

```
if xWristDistance > yWristDistance then
  elbowPos = 180 * xWristDistance / MaxWristRange + (90-rotatePos)/2
else
  elbowPos = 180 * yWristDistance / MaxWristRange + (90-rotatePos)/2
```

Algorithm 4 Servo Timer

```
if clock then
  if clockCount = clockDiv then
    clockTick = 1
  else
    clockTick = 0
  if clockTick = 1 then
    clockCount = 0
  else
    clockCount +=1
```

Algorithm 5 Servo Driver

```
if clock then
  if clockTick is 1 then
    pulseCount +=1
    if pulseCount < servoPos+47 then
      servoPin = 1
    else
      servoPin = 0
```

Algorithm 6 Clamp Algorithm

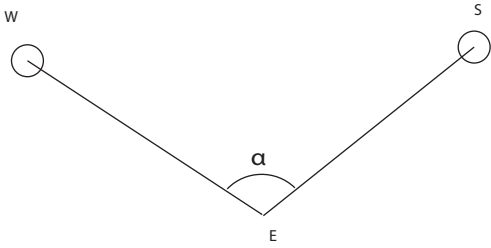
```
if thumb is found then
  if abs(xWrist - Xthumb) + abs(yWrist - Ythumb)<150 then
    handPos = 90
  else
    handPos = 180
```

Algorithm 7 Special Case

```
if frames from the front case < 75 then  
    rotatePos = 0  
    elbowPos = 5 * handPixelCount / 128  
    shoulderPos = 3*yHand/8
```

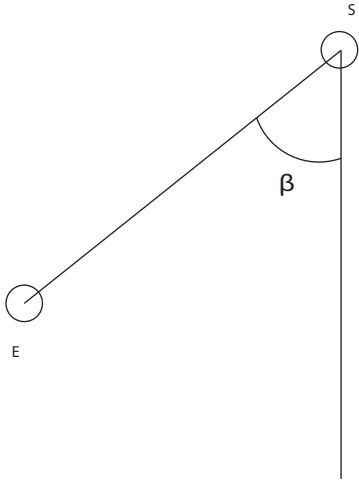
Algorithm 8 Generic Marker Tracking

```
if calibrateMarker = 1 then  
    if pixelCount = 100 and lineCount = 100 then  
        RMarker = Rout  
        GMarker = Gout  
        BMarker = Bout  
        Xmarker = 100  
        Ymarker = 100  
        markerCalibrated = 1  
    foundMarkerLastFrame = foundMarker  
    if foundMarker = 1 then  
        XMarkerLastFrame = foundXMarker  
        YMarkerLastFrame = foundYMarker  
    foundMarker = 0  
    diffCurMarker = Rout - Rmarker + Gout - Gmarker + Bout-Bmarker  
    if foundMarkerLastFrame = 1 then  
        diffCurMarker = diffCurMarker + XmarkerLastFrame - pixelCount +  
        (YmarkerLastFrame - linecount)/2  
    if calibratedMarker = 1 and foundMarker = 0 and diffCurMarker ; thresh-  
old OR foundMarker = 1 AND diffCurMarker < diffBestMarker then  
        foundMarker = 1  
        foundXMarker = pixelCount  
        foundYMarker = lineCount  
        diffBestMarker = diffCurMarker
```



$$\cos(\alpha) = \frac{(W.x - E.x)^2 + (W.y - E.y)^2 + (E.x - S.x)^2 + (E.y - S.y)^2 - (W.x - S.x)^2 - (W.y - S.y)^2}{2\sqrt{(W.x - E.x)^2 + (W.y - E.y)^2}\sqrt{(E.x - S.x)^2 + (E.y - S.y)^2}}$$

Figure 8: Elbow Movement



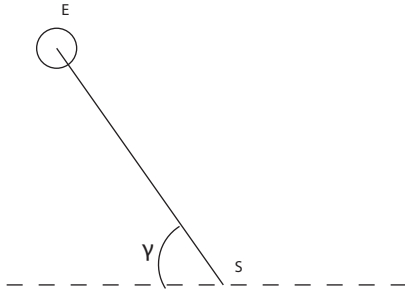
$$\cos(\beta) = \frac{(E.x - S.x)^2 + (E.y - S.y)^2 + (S.y - E.y)^2 - (S.x - E.x)^2}{2\sqrt{(E.x - S.x)^2 + (E.y - S.y)^2}(S.y - E.y)}$$

Figure 9: Shoulder Y

Questions

Briefly describe the functionality of each module along with their interconnection infrastructure (both the sub-modules within video_capture and all the modules implemented in the FPGA including PLB/OPB buses

video_capture sub-modules: Thumb/Hand/Wrist/Elbow/Shoulder Marker Detection: Based on object tracking techniques, finds and records the X and Y



$$\cos(\gamma) = \frac{(E.x - S.x)^2 + (E.y - S.y)^2 + (S.y - E.y)^2 - (S.x - E.x)^2}{2\sqrt{(E.x - S.x)^2 + (E.y - S.y)^2}(S.x - E.x)}$$

Figure 10: Shoulder X

location of our markers. Arm Angle Calculation: Using the X and Y signals updated by our marker detection, this sub-module calculates the degree at which our servos should be for accurate ICRA movement. Servo Control (Hand; Wrist; Elbow; Shoulder; Rotate): Controls the degree at which each of our servos should be turned at any given time.

What assumptions you added/modified compared to your original proposal and why

video_capture sub-modules: - Originally, we said wed use complex math to calculate our arm angles. Due to FPGA timing issues, we found wed have to reduce our frame sampling time to attempt to utilize it. When faced with this trade-off, we opted for rough approximations over more accurate calculations to maintain our real-time ICRA movement. - In our proposal, we make mention of out-of-bound conditions for the camera without specially stating what they all are. In order for the program to properly calculate the arms position, it became necessary to define a range from the camera where the user gets optimal ICRA movement. The farther the user is from the optimal range, the less sensitive ICRA is to the users own movement. This assumption had to be made for our rough approximations to work more accurately.

List the optimization techniques you came up with to improve the performance and design

All the axis movements had to be adjusted with optimization techniques. Traditional mathematical algorithms were too complex to be computed with in real time given the limitations of the FPGA hardware. Timing issues forced heuristics to be formed for all calculations...

List any interesting observations you had

More complex or correct calculations could be made with faster hardware. Further testing could reveal a better optimization code for more accurate image processing. A better image processing algorithm for more accurate tracking would be useful. Smaller tracking devices would allow more accurate calculations to be made.

List how many slices are used by each module in the FPGA

Module	Slice Count
system	3889
ppc405_0_wrapper	0
ppc405_1_wrapper	0
jtagppc_0_wrapper	2
reset_block_wrapper	31
plb_wrapper	241
opb_wrapper	48
plb2opb_wrapper	509
rs232_uart_1_wrapper	54
plb_bram_if_cntlr_1_wrapper	196
plb_bram_if_cntlr_1_bram_wrapper	0
dcm_0_wrapper	2
i2c_wrapper	284
video_capture_0_wrapper	2204
reset_split_wrapper	0
and_gate_0_wrapper	1
i2c_rst_or_wrapper	1
video_mung_wrapper	42

Figure 11: Module and Slice Count

Name	Module	Function
ppc405	PPC405	Instantiates PowerPC 405 Processor Block.
opb	On-Chip Peripheral Bus V2.0 with OPB Arbiter	Its used as an interconnect for the FPGA embedded processor systems.
plb	Processor Local Bus	Consists of bus control unit, watchdog timer, separate address, write, and read data path units with a three-cycle only arbitration feature. Contains DCR slave interface to provide access to bus error status registers and power-up reset circuit to ensure a PLB reset is generated.
plb_bram_if	PLB Block RAM (BRAM) Interface Controller	Attaches to the PLB and is the interface between the PLB and the bram.block peripheral.
dcm	Digital Clock Manager (DCM) Module	A wrapper around the DCM primitive which allows for use in the EDK tool suite. Used to implement delay locked loop, digital frequency synthesizer, digital phase shifter, or a digital spread spectrum.
plb2opb	PLB to OPB Bridge	Translates PLB transactions into OPB transactions. Functions as a slave on PLB side and a master on OPB side. Necessary in systems where a PLB master device requires access to OPB peripherals.
jtagppc	JTAGPPC Controller	A wrapper for the JTAGPPC FPGA primitive. Allows the PowerPC to connect to the JTAG chain of the FPGA.
reset_block	Processor System Reset Module	Allows the customer to tailor the design to suit their application by setting certain parameters to enable/disable features.
reset_split	Reset Module Bus Split	Provides a reset output signal to two different modules.
RS232_Uart	OPB UART Lite	A module that attaches to the OPB.

Figure 12: FPGA Modules

Name	Module	Function
i2c	OPB IIC Bus Interface	Module that provides easy-to-use interface for the I2C bus. Bus consists of a serial data (SDA) wire and a serial clock (SCL) wire and allows for START and STOP signal generation/detection.
i2c_rst_or	I2C OR gate	Provides expected OR gate logic and functionality to the I2C module.
and_gate	AND gate	Provides expected AND gate logic and functionality.
video_capture	Video Capture	Modification & Handles the video input from the camera and provides it (possibly modified as well) to any visual outputs that would use it.
opb_gpio	OPB General Purpose Input/Output	Provides a general purpose I/O interface to a 32-bit On-Chip Peripheral Bus and can be configured as either a single or a dual channel device.

Figure 13: FPGA Modules Cont.