Hardware implementation of the tiff algorithm

User manual

Date: *January 7, 2014*

By:
Aart Mulder

Abstract

This document considers the hardware implementation of the b/w tiff compression algorithm(CCITT-G4). Firstly it explains how to setup the project, run the testbench and build the implementation. Secondly it explains elements that the design exists of and usage of the client application. The advantage of this design is a very low bandwith and low energy consumption in situations where a remote camera is needed that possibly runs on a battery.

Contents

Chapter 1

Introduction

This document explains the functionality of the CCITT-G4 hardware implementation. This vhdl design works in combination with a client application running on a PC that handles and displays tiff images received over RS232 from an FPGA.

Future extensions:

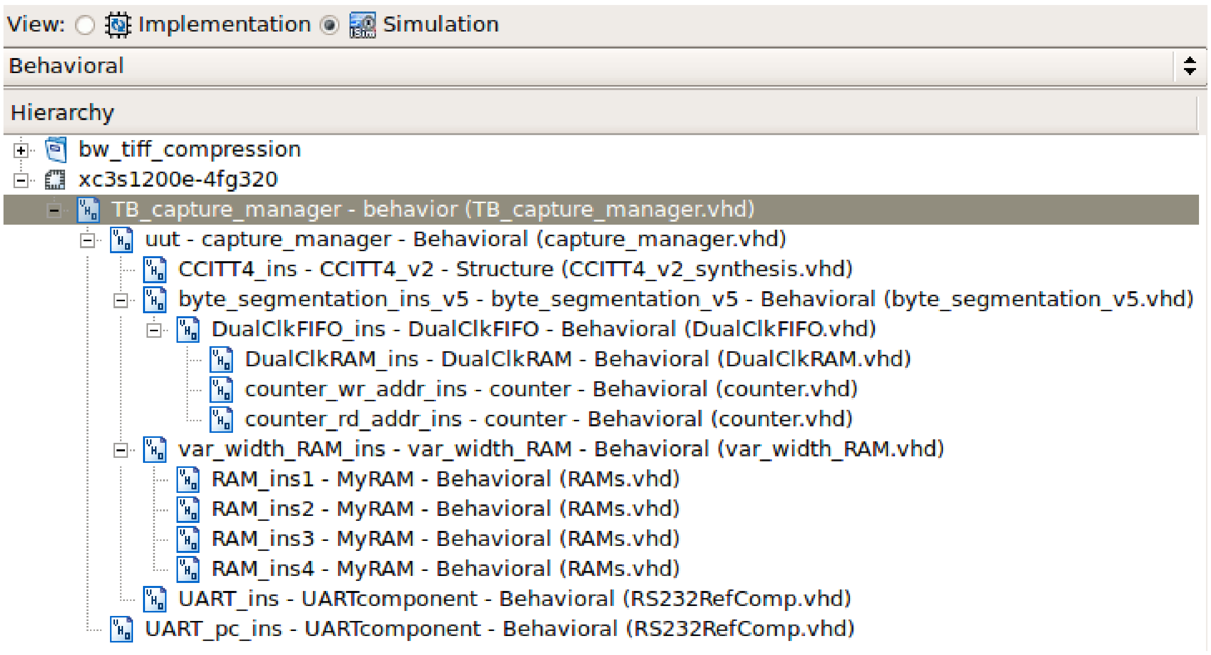
Region of interest coding.

Change coding of region of interest.

Chapter 2

Project setup

The complete project directory that can be downloaded or checked out using svn contains a sub-directory called *prj* that contains the Xilinx ISE project file *bw_tiff_compression.xise*.



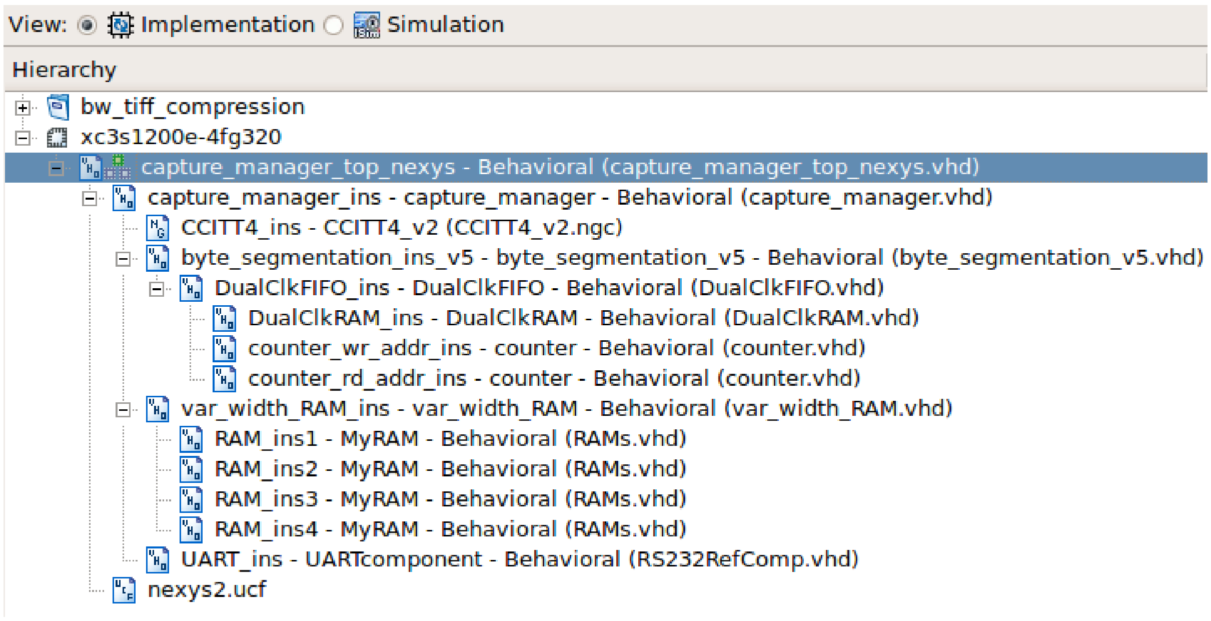
View: ○ Implementation ● Simulation

Behavioral

Hierarchy

```
bw_tiff_compression
xc3s1200e-4fg320
  TB_capture_manager - behavior (TB_capture_manager.vhd)
    uut - capture_manager - Behavioral (capture_manager.vhd)
      CCITT4_ins - CCITT4_v2 - Structure (CCITT4_v2_synthesis.vhd)
      byte_segmentation_ins_v5 - byte_segmentation_v5 - Behavioral (byte_segmentation_v5.vhd)
        DualClkFIFO_ins - DualClkFIFO - Behavioral (DualClkFIFO.vhd)
          DualClkRAM_ins - DualClkRAM - Behavioral (DualClkRAM.vhd)
          counter_wr_addr_ins - counter - Behavioral (counter.vhd)
          counter_rd_addr_ins - counter - Behavioral (counter.vhd)
      var_width_RAM_ins - var_width_RAM - Behavioral (var_width_RAM.vhd)
        RAM_ins1 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins2 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins3 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins4 - MyRAM - Behavioral (RAMs.vhd)
      UART_ins - UARTcomponent - Behavioral (RS232RefComp.vhd)
    UART_pc_ins - UARTcomponent - Behavioral (RS232RefComp.vhd)
```

Figure 2.1: Xilinx ISE project explorer in simulation mode.


View: ● Implementation ○ Simulation

Hierarchy

```
bw_tiff_compression
xc3s1200e-4fg320
  capture_manager_top_nexys - Behavioral (capture_manager_top_nexys.vhd)
    capture_manager_ins - capture_manager - Behavioral (capture_manager.vhd)
      CCITT4_ins - CCITT4_v2 (CCITT4_v2.ngc)
      byte_segmentation_ins_v5 - byte_segmentation_v5 - Behavioral (byte_segmentation_v5.vhd)
        DualClkFIFO_ins - DualClkFIFO - Behavioral (DualClkFIFO.vhd)
          DualClkRAM_ins - DualClkRAM - Behavioral (DualClkRAM.vhd)
          counter_wr_addr_ins - counter - Behavioral (counter.vhd)
          counter_rd_addr_ins - counter - Behavioral (counter.vhd)
      var_width_RAM_ins - var_width_RAM - Behavioral (var_width_RAM.vhd)
        RAM_ins1 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins2 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins3 - MyRAM - Behavioral (RAMs.vhd)
        RAM_ins4 - MyRAM - Behavioral (RAMs.vhd)
      UART_ins - UARTcomponent - Behavioral (RS232RefComp.vhd)
    nexys2.ucf
```

Figure 2.2: Xilinx ISE project explorer in implementation mode.

2.1 Simulation

2.2 Synthesis

Chapter 3

Implementation

Figure 3.1 shows the block diagram of the complete system with the camera, fpga and PC. The single modules are explained in the sections that follow.

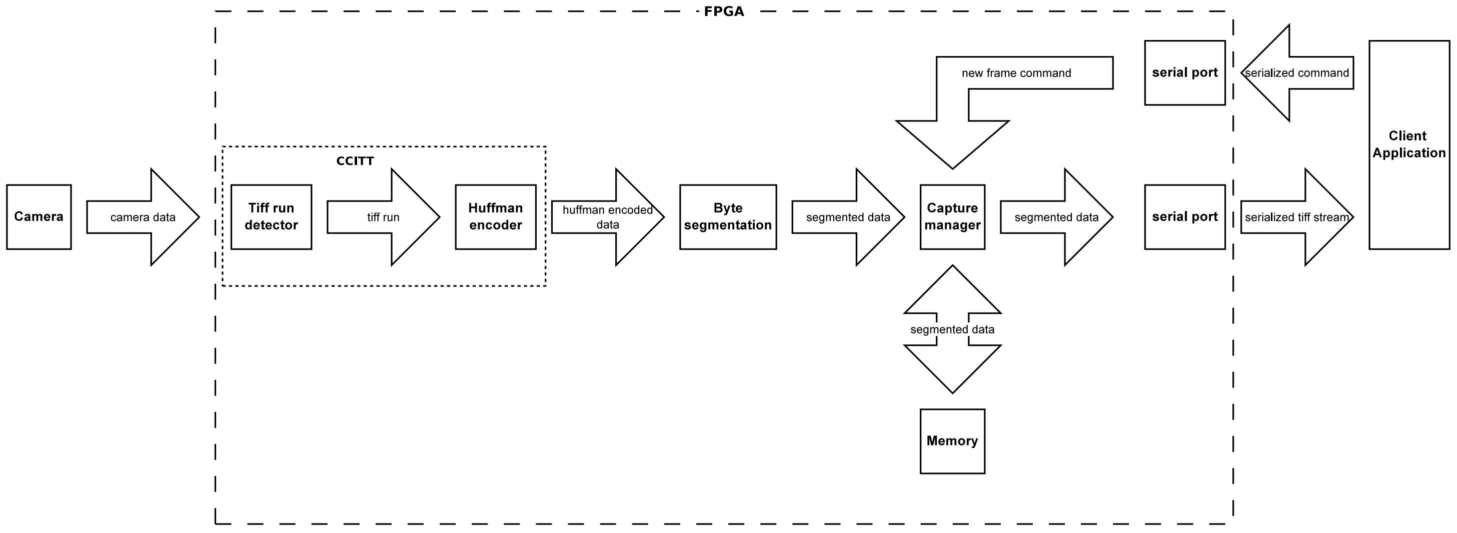


Figure 3.1: Dataflow through the system

3.1 CCITT

The pixels entering the system from the camera are first processed by the Tiff-run detector that does change coding based on the current and previous line. The output of the Tiff-run detector is fed into the Huffman encoder which applies huffman encoding based on table A.1 and A.2 in appendix A. Those two modules together do the actual image compression and are therefore called the CCITT module. The flow diagram in figure 3.2 comes from the CCITT-G4 standard and shows how the Tiff-run detector works.

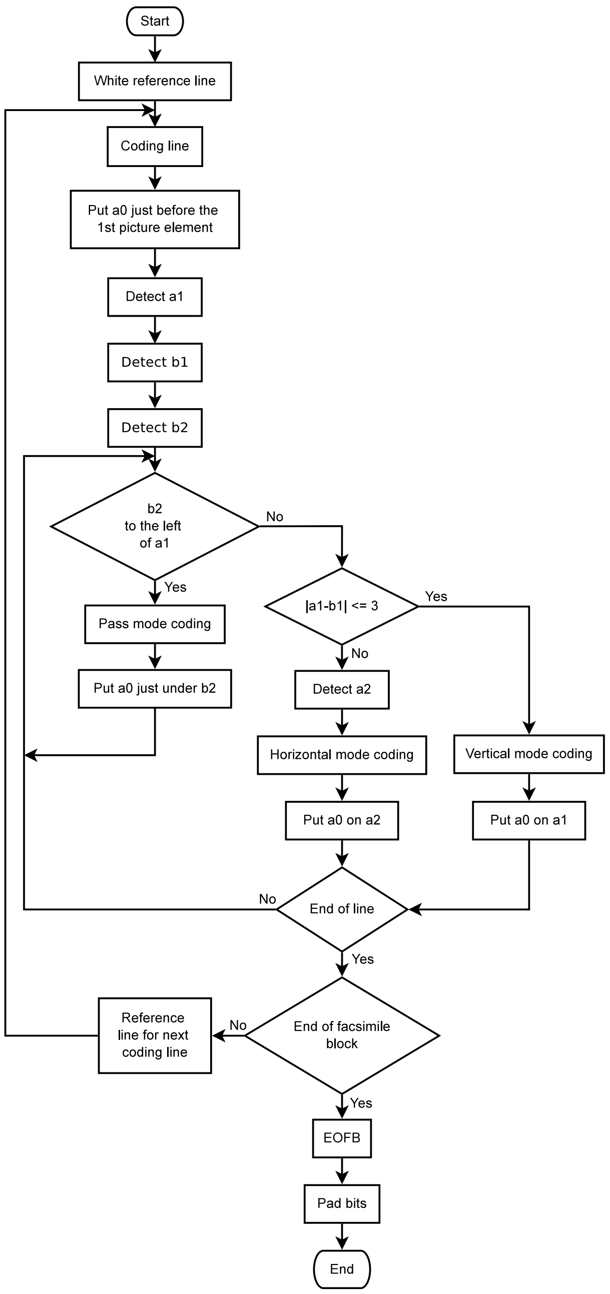


Figure 3.2: ITU-T Rec. T.6. coding flow diagram
[1]

3.2 Byte segmentation

The byte segmentation module does as it's name suggests, segmentation of the input data, coming from the CCITT module into byte segments. A symbolic diagram shown in figure 3.3. The input data is buffered in the FIFO to handle sudden bursts. Theoretically it is possible that several bigger huffman encoded segments come in without any clock cycle in between but at the same time image patterns that change on every pixel have a small segment size. Depending on the data size available in the shift register output 2,3 and 4 are used as well to empty the register as fast as possible. The data left over after writing to the output is moved up to the top.

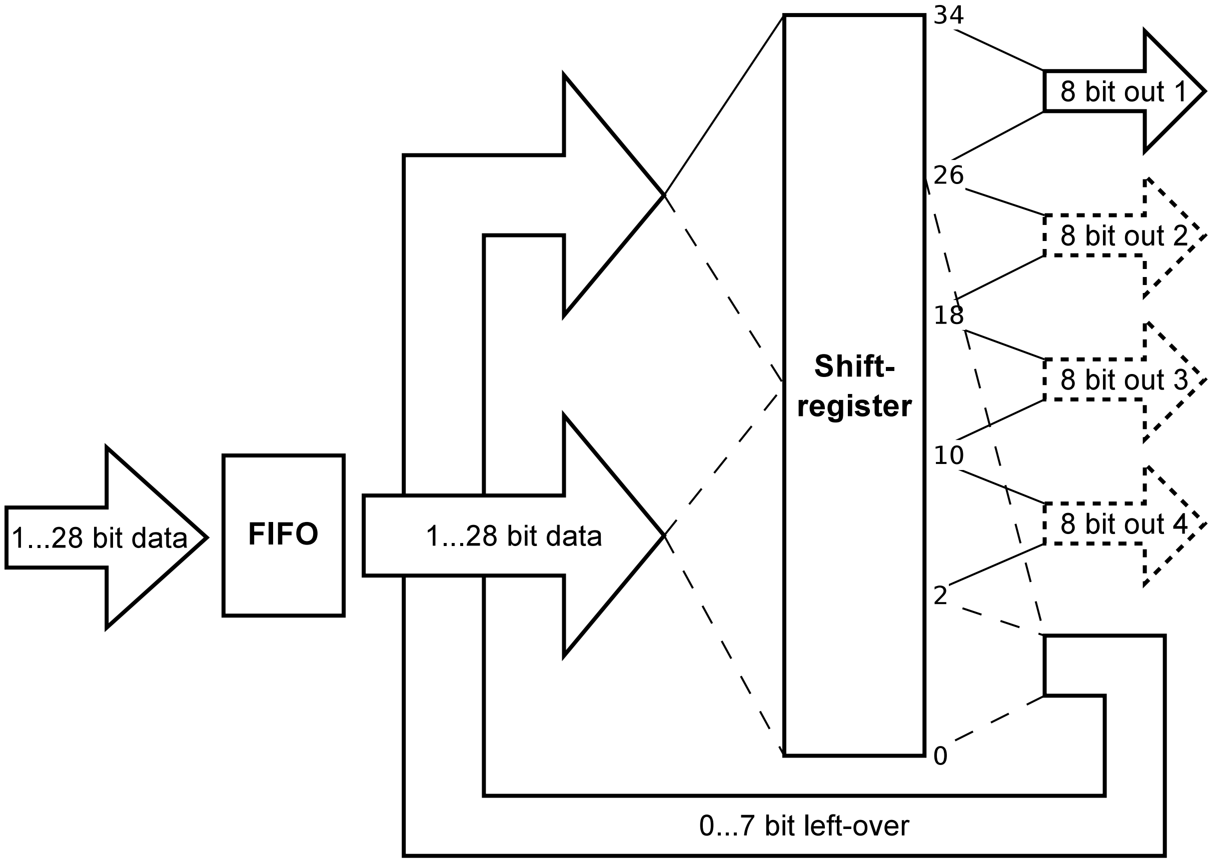


Figure 3.3: Block diagram of the byte segmentation module.

3.3 Transmission memory

The four 8 bit data signals coming from the byte segmentation module are stored in the transmission memory. Before storage the data goes through a kind of rotating register, drawn as the big circle in figure 3.4. When the first data of 8 bit comes in, the multiplexer passes it through to position 1 in RAM 1. Then when the second data of for example 32 bit comes in, the multiplexer routes byte 1 to RAM 2, byte 2 to RAM 3, byte 3 to RAM 4 and byte 4 to position 2 of RAM 1. On data read the four RAMs are read sequentially, i.e. first position 1 of all the RAMs and then position 2 of all the RAMs and so on. A detailed block diagram is shown in appedix B.

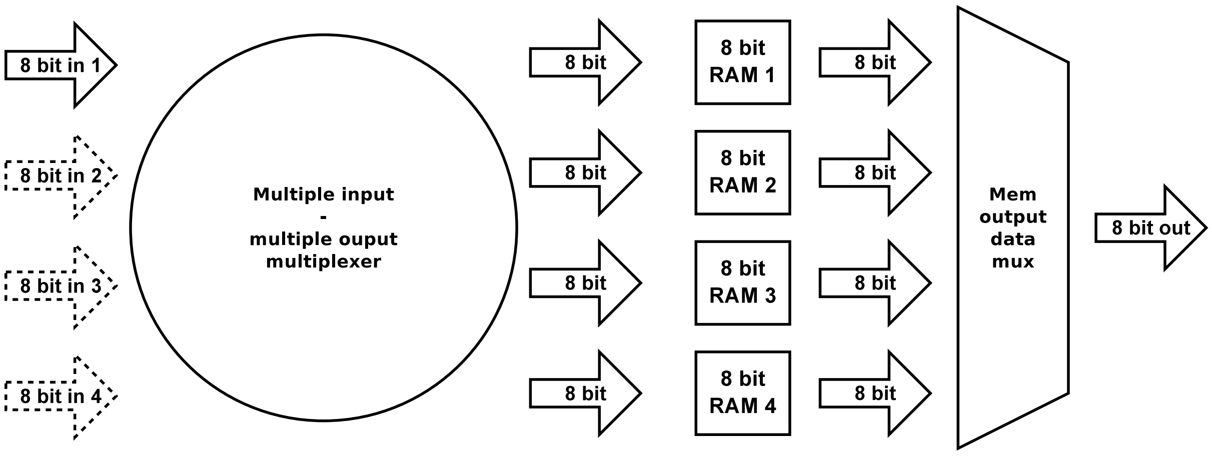


Figure 3.4: Variable with input RAM module.

3.4 Capture manager

The task of the capture manager is to handle input commands received over RS232 and to control the capturing process. It waits for a new-frame command, then captures and stores the next complete frame and sends it out over the RS232 connection. Figure 3.5 shows the state machine that controls the system.

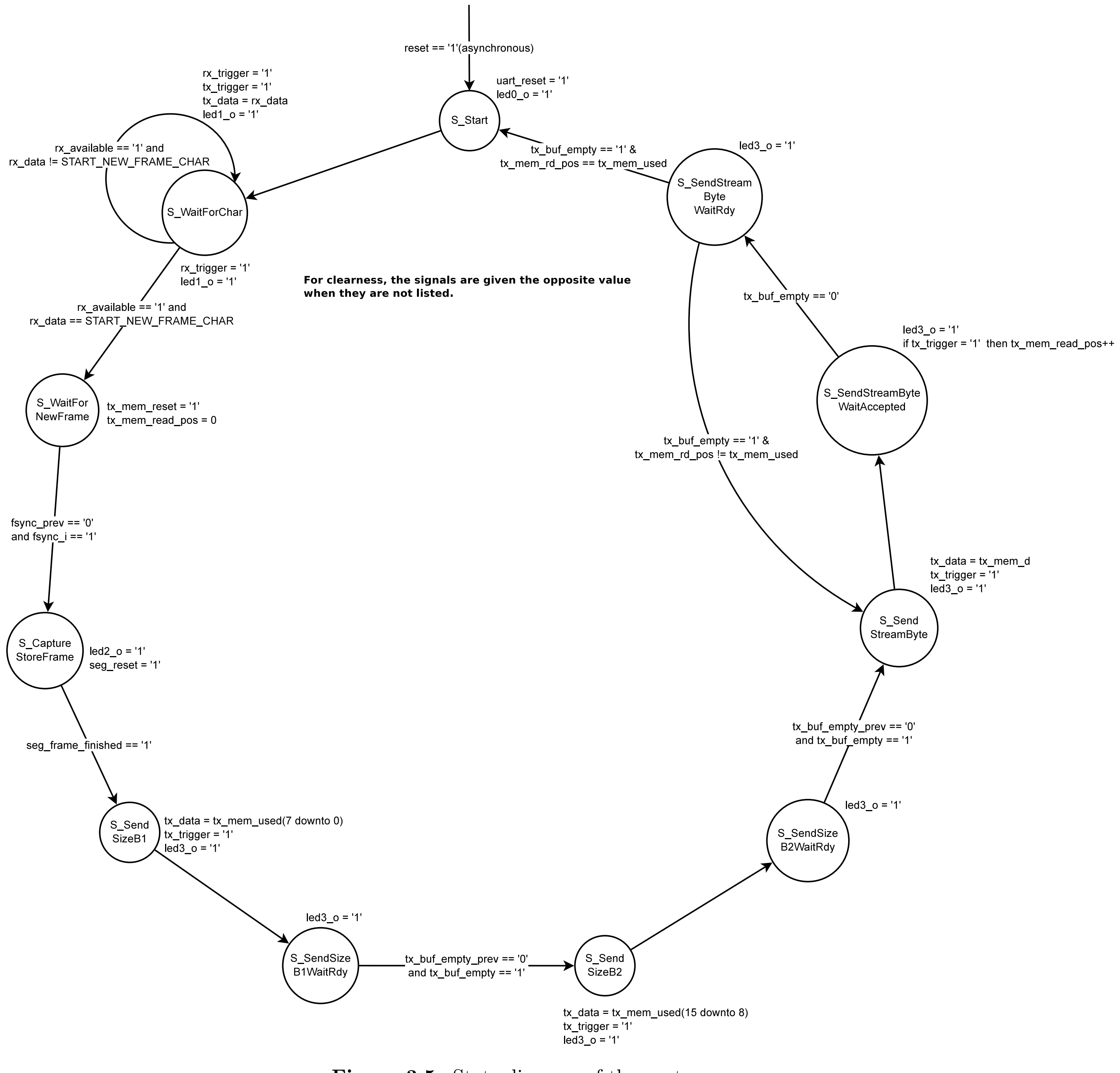


Figure 3.5: State diagram of the capture manager.

3.5 Serial communication

The serial port is used for communication with the client application on the PC that stores the tiff streams. It receives commands from the PC and sends the tiff stream to the PC. A header with the tiff stream size is send before the stream itself to let the PC know how much data to expect. Another option is to use timeouts but that would limit the throughput since the timeout must be longer than the longenst expectable stream. The serial port component is obtained from Digilent, the developer of the Nexys2 FPGA board used for testing the design. The baudrate can be set based on the clock speed. The maximum functional baudrate in my situation was 115200 baud at a clock speed of 26.67MHz. A higher resulted in communication failures.

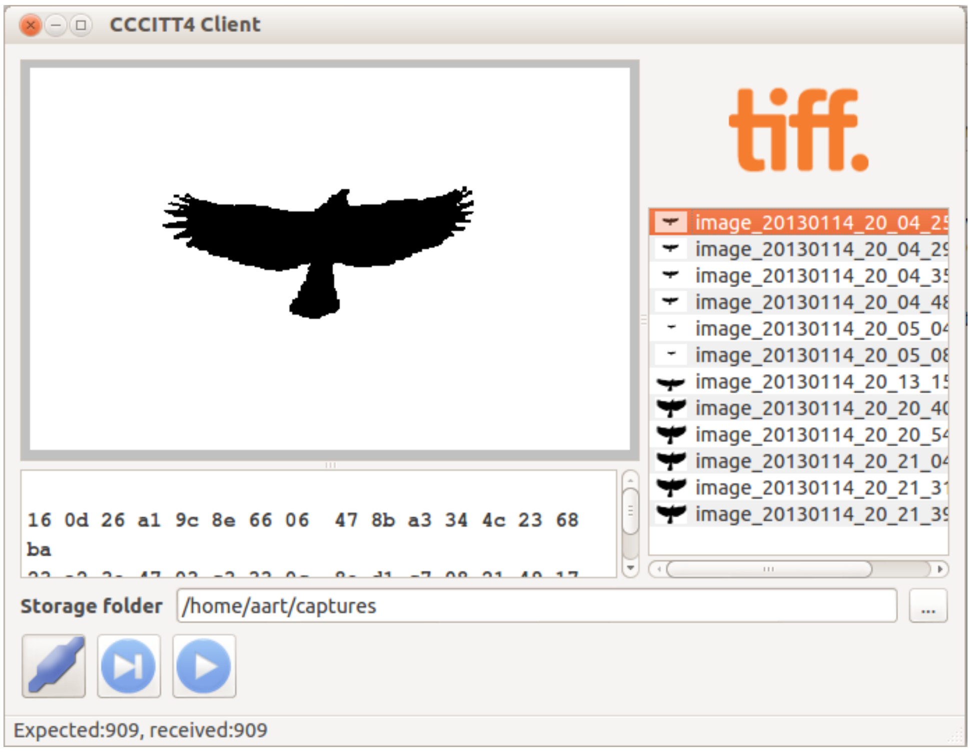
Chapter 4

Client application

The client application as described in the previous chapter is shown below in figure 4.1. It is a fairly simple application that connects to the serial port, sends a new-frame command out and waits for data to be stored in a tiff file. When a complete image is received, it is displayed in the image viewer. Images stored in the same folder are listed in the panel in the right side. The storage folder can be changed by typing the path or with the button on the right that opens a selection window. The text window below the image viewer is a receiving data debug log. The three buttons at the bottom left are respectively connect/disconnect, single mode and continuous mode. The connect/disconnect button opens the port configuration window shown in figure 4.2. The application contains a timeout timer of currently 10 seconds to cancel the communication if the server(fpga) doesn't reply quick enough. The communication can be canceled at any time by pressing the Esc key.

The project sub folder *client_application/target* contains two executables for respectively Windows and Ubuntu which are build in respectively 32 and 64 bit format but have been executed successfully in Windows7 x64. Furthermore it contains the **.dll* files that are needed to run the application in Windows. The sub folder *qt* must be placed in the *C* directory. It contains the runtime library *qtiff.dll* that is needed to open tif. This file must be stored under the exact path *c:\qt\4.8.1\desktop\4.8.1\mingw\plugins\imageformats* because the application appears to have a static path to it. On Ubuntu it runs straight away, at least in a clean install of 13.10.

The source can be build if one has a version of Qt installed, so far only tested with 4.8 but according to the Qt community any version should work. Version 5 contains some major changes so I'm not sure if that works. On Linux/Ubuntu the standard gcc compiler is used to build but on Windows MinGW or Visual Studio must be installed. The operating system specific code parts for the serial port are selected based on preprocessor defines and no odd libraries are used for it.



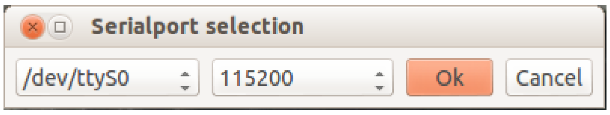
```
image_20130114_20_04_25
image_20130114_20_04_29
image_20130114_20_04_35
image_20130114_20_04_48
image_20130114_20_05_04
image_20130114_20_05_08
image_20130114_20_13_15
image_20130114_20_20_40
image_20130114_20_20_54
image_20130114_20_21_04
image_20130114_20_21_31
image_20130114_20_21_39
```

```
16 0d 26 a1 9c 8e 66 06  47 8b a3 34 4c 23 68
ba
```

Storage folder /home/aart/captures

Expected:909, received:909

Figure 4.1: The client application.



/dev/ttyS0 115200

Figure 4.2: The port selection window.

Appendix

Appendix A

Huffman tables

White run length	Code word	Black run length	Code word
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
320	00110110	320	000000110011
384	00110111	384	000000110100
448	01100100	448	000000110101
512	01100101	512	0000001101100
576	01101000	576	0000001101101
640	01100111	640	0000001001010
704	011001100	704	0000001001011
768	011001101	768	0000001001100
832	011010010	832	0000001001101
896	011010011	896	0000001110010
960	011010100	960	0000001110011
1024	011010101	1024	0000001110100
1088	011010110	1088	0000001110101
1152	011010111	1152	0000001110110
1216	011011000	1216	0000001110111
1280	011011001	1280	0000001010010
1344	011011010	1344	0000001010011
1408	011011011	1408	0000001010100
1472	010011000	1472	0000001010101
1536	010011001	1536	0000001011010
1600	010011010	1600	0000001011011
1664	011000	1664	0000001100100
1728	010011011	1728	0000001100101

Table A.1: Make-up codes.
[1]

White run length	Code word	Black run length	Code word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	0000010111
17	101011	17	0000011000
18	0100111	18	0000001000
19	0001100	19	00001100111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	00101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	01011001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

Table A.2: Termination codes.
[1]

Appendix B

Detailed transmission memory block diagram

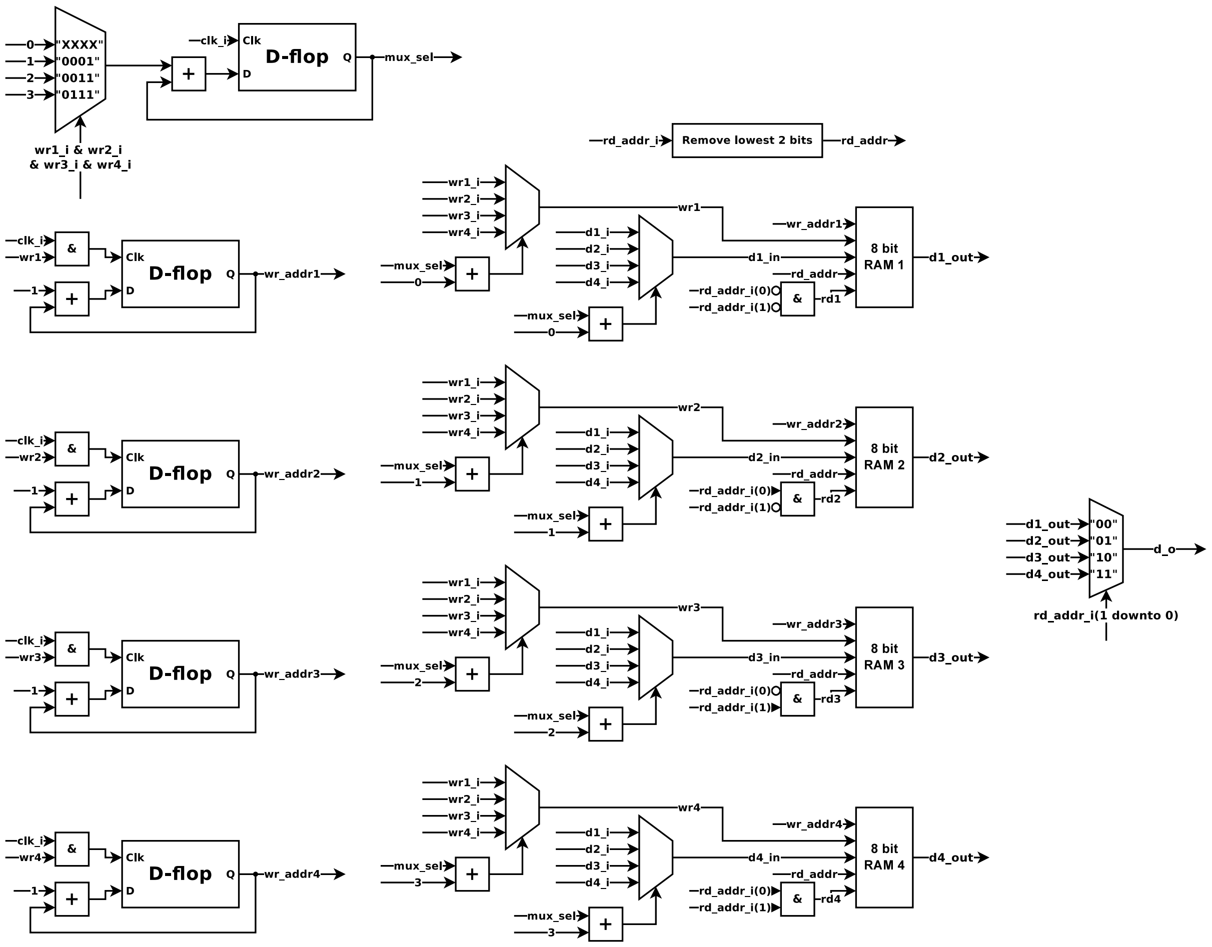


Figure B.1: Detaile block diagram of the variable with input RAM module.

Bibliography

[1] `http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.6-198811-I!!PDF-E&type=items`. Itu-t (ccitt) t.6. facsimile coding schemes and coding control functions for group 4 facsimile apparatus, 6 December, 2012.