Appendix A

This section of the appendix will list the Handel-C code for the WMSN FPGA processor implementing the DWT.

```
// Handel-C Code for WMSN FPGA Processor Implementing the DWT

// === Pre-processor includes === //
#define PAL_TARGET_CLOCK_RATE 37500000
#include "pal_master.hch"
#include "stdlib.hch"

// === Parameter initializations === //
macro expr ClockRate        = PAL_ACTUAL_CLOCK_RATE;

macro expr ImageWidth       = 128;
macro expr ImageHeight      = 128;
macro expr StripHeight      = 16;
macro expr PatchWidth       = 8; //8;
macro expr PatchHeight      = 8; //8;
macro expr ImageBits        = 6;
macro expr StripBits        = 16;   // half of the standard 32-bits

macro expr NumStrips        = ImageHeight / StripHeight;
macro expr NumPatches       = (ImageWidth * StripHeight) / (PatchWidth *
     PatchHeight);

macro expr PatchesPerRow    = StripHeight / PatchHeight;
macro expr PatchesPerCol    = ImageWidth / PatchHeight;

macro expr RAMImageAddress  = ImageWidth * ImageHeight;
macro expr RAMStripAddress  = 8192;
macro expr InstMemSize      = 2048; // Allocated instruction memory size

macro expr RAMImageAddressBits = log2ceil(RAMImageAddress);
macro expr RAMStripAddressBits = log2ceil(RAMStripAddress);

macro expr StripCountBits      = log2ceil(NumStrips) + 1;
macro expr PatchCountBits      = log2ceil(NumPatches) + 1;

macro expr ImagePixelCountBits = RAMImageAddressBits + 1;
macro expr StripPixelCountBits = RAMStripAddressBits - 2 + 1;
macro expr PatchPixelCountBits = log2ceil(PatchWidth * StripHeight) + 1;
macro expr PatchWidthCountBits = log2ceil(PatchWidth) + 1;

macro proc ReadCamera();
macro proc LoadStripMemory();
macro proc RunProcessor();
```

L. Ang et al., *Wireless Multimedia Sensor Networks on Reconfigurable Hardware*,
DOI 10.1007/978-3-642-38203-1, © Springer-Verlag Berlin Heidelberg 2013

```
macro proc OutputStripMemory();

// Component declarations

ram unsigned int ImageBits IMAGE_MEMORY[RAMImageAddress] with {block = 1};
ram unsigned int StripBits STRIP_MEMORY[RAMStripAddress] with {block = 1};


unsigned int RAMImageAddressBits RAMAddressImageMemory;
unsigned int RAMStripAddressBits RAMAddressStripMemory;

unsigned int StripCountBits StripCounter;

void main (void)
{
    par
    {   RC10LEDWriteMask(15);               // Light up four LSB LEDs to
         indicate start of processing
        // Run PAL/PSL functions
        RC10CameraRun(OV9650_RGB565_QVGA_LowLight,ClockRate/2);     //
             camera runs at 12MHz
        RC10RS232Run (RC10RS232_9600Baud,RC10RS232ParityNone,
             RC10RS232FlowControlHard,ClockRate);
        seq
        {
            ReadCamera();
            RAMAddressImageMemory = 0;
            StripCounter = 0;

            while (StripCounter != NumStrips)
            {
                LoadStripMemory();
                RunProcessor();
                OutputStripMemory();
                StripCounter++;
            }

            RC10LEDWriteMask(255);      // Light up remaining four LEDs to
                 indicate all functions have been performed
        }
    }

} // end main

static macro proc ReadCamera()
{
    unsigned int CameraAddressX;
    unsigned int CameraAddressY;
    unsigned int CameraPixel;
    unsigned int ImagePixelCountBits ImagePixelCounter;
    unsigned int RAMImageAddressBits ImageAddress;

    seq
    {
        RAMAddressImageMemory = 0;
        ImagePixelCounter     = 0;
        while (ImagePixelCounter != RAMImageAddress)
        {
            par
            {
                IMAGE_MEMORY[RAMAddressImageMemory] = 0;
                RAMAddressImageMemory++;
                ImagePixelCounter++;
            }
        }
        do
        {
```

```
            RC10CameraReadRGB565(&CameraAddressX,&CameraAddressY,&
                 CameraPixel);
        } while (!(CameraAddressX == 96 && CameraAddressY == 56));

        ImageAddress = 0;

        // Take the center portion of the 240x320 capture
        do
  {
   par
            {
                RC10CameraReadRGB565(&CameraAddressX,&CameraAddressY,&
                     CameraPixel);
                if (CameraAddressX <= 223 && CameraAddressX >= 96)
                {
                    par
                    {
                        IMAGE_MEMORY[ImageAddress] = 0@CameraPixel[10:5];
                        ImageAddress++;
                    }
                }
                else
                {
                   delay;
                }
            }
  } while (CameraAddressY != 184);
    }
}

static macro proc LoadStripMemory()
{
    unsigned int StripPixelCountBits StripPixelCounter;
    unsigned int PatchCountBits      PatchCounter;
    unsigned int ImageBits           WriteData;
    seq
    {
        RAMAddressStripMemory = 0;
        StripPixelCounter     = 0;
        WriteData             = 0@IMAGE_MEMORY[RAMAddressImageMemory];
        RAMAddressImageMemory++;
        while (StripPixelCounter != (ImageWidth * StripHeight))
        {
            par
            {
                WriteData = IMAGE_MEMORY[RAMAddressImageMemory];
                STRIP_MEMORY[RAMAddressStripMemory] = 0@WriteData;
                RAMAddressImageMemory++;
                RAMAddressStripMemory++;
                StripPixelCounter++;
            }
        }
        // Initialize strip patches to valid
        RAMAddressImageMemory--;
        RAMAddressStripMemory = RAMStripAddress - NumPatches;
        PatchCounter = 0;
        while (PatchCounter != NumPatches)
        {
            par
            {
                STRIP_MEMORY[RAMAddressStripMemory] = 1;
                RAMAddressStripMemory++;
                PatchCounter++;
            }
        }
    }
}
```

```
static macro proc RunProcessor()
{
    // === Instruction Memory === //
    static ram unsigned int 32 INST_MEMORY[InstMemSize] =
    {
0b00100000000000011000000000000000, L0. ADDI $R0 $R3 0  \\ Start of
     Algorithm 4.11
0b00100000000000010000010000000000, L1. ADDI $R0 $R4 4096
0b00100000000000010100000100000000, L2. ADDI $R0 $R5 2048
0b00100000000000011000000000000000, L3. ADDI $R0 $R6 0
0b00000000011000101101100000101010, L4. SLT $R6 $R5 $R22 \\ while $R6 < $R5
     do
0b00010110110000010000000000001110, L5. BNE $R22 $R1 14 \\ branch to line 14
      if the condition above is not true
0b00000000000000000000000000000000, L6. SLL $R0 $R0 $R0
0b10001100011001110000000000000000, L7. LW $R3 $R7 0
0b00000000011000010001100000100000, L8. ADD $R3 $R1 $R3
0b10101100100001110000000000000000, L9. SW $R4 $R7 0
0b00000000100000010010000000100000, L10. ADD $R4 $R1 $R4
0b00000000110000010011000000100000, L11. ADD $R6 $R1 $R6
0b00010000000000000000000000000100, L12. BEQ $R0 $R0 4
0b00000000000000000000000000000000, L13. SLL $R0 $R0 $R0 \\ End of Algorithm
     4.11
0b00100000000000011000000000010000, L14. ADDI $R0 $R3 16 \\ Start of
     Algorithm 4.12
0b00100000000001000000000010000000, L15. ADDI $R0 $R4 128
0b00100000000001010000100000000000, L16. ADDI $R0 $R5 2048
0b00100000000001100000000000000001, L17. ADDI $R0 $R6 1
0b00100000000001110001000000000000, L18. ADDI $R0 $R7 4096
0b00100000000010000001000000000000, L19. ADDI $R0 $R8 4096
0b00000000000000001000100000100000, L20. ADD $R0 $R0 $R17
0b00000000000000001001000000100000, L21. ADD $R0 $R0 $R18
0b00000000000000001001100000100000, L22. ADD $R0 $R0 $R19
0b00000000000000001010000000100000, L23. ADD $R0 $R0 $R20
0b00000000000000001001000000100000, L24. ADD $R0 $R0 $R9
0b00000000000000001010000000100000, L25. ADD $R0 $R0 $R10
0b00000000100000101000000000100010, L26. SUB $R4 $R2 $R16
0b00000001001000111011000000101010, L27. SLT $R9 $R3 $R22 \\ while $R9 < $R3
      do
0b00010100001101100000000001010111, L28. BNE $R1 $R22 87  \\ branch to line
     87 if the condition above is not true
0b00000000000000000000000000000000, L29. SLL $R0 $R0 $R0
0b00000001010001001011000000101010, L30. SLT $R10 $R4 $R22 \\ while $R10 <
     $R4 do
0b00010110110000010000000001010011, L31. BNE $R22 $R1 83 \\ branch to line
     83 if the condition above is not true
0b00000000000000000000000000000000, L32. SLL $R0 $R0 $R0
0b00010101010100000000000000101001, L33. BNE $R10 $R16 41
0b00000000000000000000000000000000, L34. SLL $R0 $R0 $R0
0b10001100111100100000000000000000, L35. LW $R7 $R18 0  \\ Start of
     Algorithm 4.13
0b00000000111000010011100000100000, L36. ADD $R7 $R1 $R7
0b10001100111100110000000000000000, L37. LW $R7 $R19 0
0b00000000111000010011100000100000, L38. ADD $R7 $R1 $R7
0b00010000000000000000000000101110, L39. BEQ $R0 $R0 46
0b00000000000000000000000000000000, L40. SLL $R0 $R0 $R0
0b10001100111100100000000000000000, L41. LW $R7 $R18 0
0b00000000111000010011100000100000, L42. ADD $R7 $R1 $R7
0b10001100111100110000000000000000, L43. LW $R7 $R19 0
0b00000000111000010011100000100000, L44. ADD $R7 $R1 $R7
0b10001100111101000000000000000000, L45. LW $R7 $R20 0  \\ End of Algorithm
     4.13
0b00010101010000000000000000111011, L46. BNE $R10 $R0 59 \\ Start of
     Algorithm 4.14
0b00000000000000000000000000000000, L47. SLL $R0 $R0 $R0
0b00000010010101001010100000110000, L48. ADDS $R18 $R20 $R21
0b00000010011101011001100000100010, L49. SUB $R19 $R21 $R19
```

```
0b00000010011100111010100000110010, L50. ADDSS $R19 $R19 $R21
0b00000010101100101001000000100000, L51. ADD $R21 $R18 $R18
0b00000010011000001000100000100000, L52. ADD $R19 $R0 $R17
0b10101101000100100000000000000000, L53. SW $R8 $R18 0
0b00000001000000001010000000100000, L54. ADD $R8 $R1 $R8
0b10101101000100110000000000000000, L55. SW $R8 $R19 0
0b00000001000000001010000000100000, L56. ADD $R8 $R1 $R8
0b00010000000000000000000001010000, L57. BEQ $R0 $R0 80
0b00000000000000000000000000000000, L58. SLL $R0 $R0 $R0
0b00010101010100000000000001000111, L59. BNE $R10 $R16 71
0b00000000000000000000000000000000, L60. SLL $R0 $R0 $R0
0b00000010010100101010100000110000, L61. ADDS $R18 $R18 $R21
0b00000010011101011001100000100010, L62. SUB $R19 $R21 $R19
0b00000010001100111010100000110010, L63. ADDSS $R17 $R19 $R21
0b00000010101100101001000000100000, L64. ADD $R21 $R18 $R18
0b10101101000100100000000000000000, L65. SW $R8 $R18 0
0b00000001000000001010000000100000, L66. ADD $R8 $R1 $R8
0b10101101000100110000000000000000, L67. SW $R8 $R19 0
0b00000001000000001010000000100000, L68. ADD $R8 $R1 $R8
0b00010000000000000000000001010000, L69. BEQ $R0 $R0 80
0b00000000000000000000000000000000, L70. SLL $R0 $R0 $R0
0b00000010010101010010101000000110000, L71. ADDS $R18 $R20 $R21
0b00000010011101011001100000100010, L72. SUB $R19 $R21 $R19
0b00000010001100111010100000110010, L73. ADDSS $R17 $R19 $R21
0b00000010101100101001000000100000, L74. ADD $R21 $R18 $R18
0b00000010011000001000100000100000, L75. ADD $R19 $R0 $R17
0b10101101000100100000000000000000, L76. SW $R8 $R18 0
0b00000001000000001010000000100000, L77. ADD $R8 $R1 $R8
0b10101101000100110000000000000000, L78. SW $R8 $R19 0
0b00000001000000001010000000100000, L79. ADD $R8 $R1 $R8 \\ End of Algorithm
      4.14
0b00000001010000100101000000100000, L80. ADD $R10 $R2 $R10
0b00010000000000000000000000011110, L81. BEQ $R0 $R0 30 \\ end while for
     $R10 < $R4 do
0b00000000000000000000000000000000, L82. SLL $R0 $R0 $R0
0b00000000000000000101000000100000, L83. ADD $R0 $R0 $R10
0b00000001001000010100100000100000, L84. ADD $R9 $R1 $R9
0b00010000000000000000000000011011, L85. BEQ $R0 $R0 27 \\ end while for $R9
      < $R3 do
0b00000000000000000000000000000000, L86. SLL $R0 $R0 $R0
0b00100000000001110001000000000000, L87. ADDI $R0 $R7 4096
0b00100000000010000001000000000000, L88. ADDI $R0 $R8 4096
0b00100000000011000001000000000000, L89. ADDI $R0 $R12 4096
0b00100000000110100000100000000000, L90. ADDI $R0 $R26 2048
0b00000000000000000100100000100000, L91. ADD $R0 $R0 $R9
0b00000000000000000101000000100000, L92. ADD $R0 $R0 $R10
0b00000000011000101000000000100010, L93. SUB $R3 $R2 $R16
0b00000000000000001000100000100000, L94. ADD $R0 $R0 $R17
0b00000000000000001001000000100000, L95. ADD $R0 $R0 $R18
0b00000000000000001001100000100000, L96. ADD $R0 $R0 $R19
0b00000000000000001010000000100000, L97. ADD $R0 $R0 $R20
0b00000001010001001011000000101010, L98. SLT $R10 $R4 $R22 \\ while $R10 <
     $R4 do
0b00010110110000010000000010111000, L99. BNE $R22 $R1 184 \\ branch to line
     184 if the condition above is not true.
0b00000000000000000000000000000000, L100. SLL $R0 $R0 $R0
0b00000001001000111011000000101010, L101. SLT $R9 $R3 $R22 \\ while $R9 <
     $R3 do
0b00010110110000010000000010110000, L102. BNE $R22 $R1 176 \\ branch to line
      176 if the condition above is not true.
0b00000000000000000000000000000000, L103. SLL $R0 $R0 $R0
0b00010101001100000000000001101111, L104. BNE $R9 $R16 111 \\ Start of
     Algorithm 4.15
0b00000000000000000000000000000000, L105. SLL $R0 $R0 $R0
0b10001100111110010000000000000000, L106. LW $R7 $R18 0
0b00000000111001000011100000100000, L107. ADD $R7 $R4 $R7
0b10001100111110011000000000000000, L108. LW $R7 $R19 0
```

```
0b00010000000000000000000001110100, L109. BEQ $R0 $R0 116
0b00000000000000000000000000000000, L110. SLL $R0 $R0 $R0
0b10001100111110010000000000000000, L111. LW $R7 $R18 0
0b00000000111001000011100000100000, L112. ADD $R7 $R4 $R7
0b10001100111110011000000000000000, L113. LW $R7 $R19 0
0b00000000111001000011100000100000, L114. ADD $R7 $R4 $R7
0b10001100111110100000000000000000, L115. LW $R7 $R20 0 \\ End of Algorithm
    4.15
0b00010101001000000000000010001001, l116. BNE $R9 $R0 137 \\ Start of
    Algorithm 4.16
0b00000000000000000000000000000000, L117. SLL $R0 $R0 $R0
0b00000010010101001010100000110000, L118. ADDS $R18 $R20 $R21
0b00000010011101011001100000100010, L119. SUB $R19 $R21 $R19
0b00000010011100111010100000110010, L120. ADDSS $R19 $R19 $R21
0b00000010101100101001000000100000, L121. ADD $R21 $R18 $R18
0b00000010011000001000100000100000, L122. ADD $R19 $R0 $R17
0b10101101000100100000000000000000, L123. SW $R8 $R18 0
0b00000001000011000101110000100010, L124. SUB $R8 $R12 $R11 \\ Start of
    Algorithm 4.17
0b00000001011001100101100000111000, L125. DWTA $R11 $R6 $R11
0b00000001011110100101100000100000, L126. ADD $R11 $R26 $R11 \\ End of
    Algorithm 4.17
0b10101101011100100000000000000000, L127. SW $R11 $R18 0
0b00000001000001000100000000100000, L128. ADD $R8 $R4 $R8
0b10101101000100110000000000000000, L129. SW $R8 $R19 0
0b00000001000011000101110000100010, L130. SUB $R8 $R12 $R11 \\ Start of
    Algorithm 4.17
0b00000001011001100101100000111000, L131. DWTA $R11 $R6 $R11
0b00000001011110100101100000100000, L132. ADD $R11 $R26 $R11 \\ End of
    Algorithm 4.17
0b10101101011100110000000000000000, L133. SW $R11 $R19 0
0b00000001000001000100000000100000, L134. ADD $R8 $R4 $R8
0b00010000000000000000000010101101, L135. BEQ $R0 $R0 173
0b00000000000000000000000000000000, L136. SLL $R0 $R0 $R0
0b00010101001100000000000010011100, L137. BNE $R9 $R16 156
0b00000000000000000000000000000000, L138. SLL $R0 $R0 $R0
0b00000010010100101010100000110000, L139. ADDS $R18 $R18 $R21
0b00000010011101011001100000100010, L140. SUB $R19 $R21 $R19
0b00000010001100111010100000110010, L141. ADDSS $R17 $R19 $R21
0b00000010101100101001000000100000, L142. ADD $R21 $R18 $R18
0b10101101000100100000000000000000, L143. SW $R8 $R18 0
0b00000001000011000101110000100010, L144. SUB $R8 $R12 $R11 \\ Start of
    Algorithm 4.17
0b00000001011001100101100000111000, L145. DWTA $R11 $R6 $R11
0b00000001011110100101100000100000, L146. ADD $R11 $R26 $R11 \\ End of
    Algorithm 4.17
0b10101101011100100000000000000000, L147. SW $R11 $R18 0
0b00000001000001000100000000100000, L148. ADD $R8 $R4 $R8
0b10101101000100110000000000000000, L149. SW $R8 $R19 0
0b00000001000011000101110000100010, L150. SUB $R8 $R12 $R11 \\ Start of
    Algorithm 4.17
0b00000001011001100101100000111000, L151. DWTA $R11 $R6 $R11
0b00000001011110100101100000100000, L152. ADD $R11 $R26 $R11 \\ End of
    Algorithm 4.17
0b10101101011100110000000000000000, L153. SW $R11 $R19 0
0b00010000000000000000000010101101, L154. BEQ $R0 $R0 173
0b00000000000000000000000000000000, L155. SLL $R0 $R0 $R0
0b00000010010101001010100000110000, L156. ADDS $R18 $R20 $R21
0b00000010011101011001100000100010, L157. SUB $R19 $R21 $R19
0b00000010001100111010100000110010, L158. ADDSS $R17 $R19 $R21
0b00000010101100101001000000100000, L159. ADD $R21 $R18 $R18
0b00000010011000001000100000100000, L160. ADD $R19 $R0 $R17
0b10101101000100100000000000000000, L161. SW $R8 $R18 0
0b00000001000011000101110000100010, L162. SUB $R8 $R12 $R11 \\ Start of
    Algorithm 4.17
0b00000001011001100101100000111000, L163. DWTA $R11 $R6 $R11
```

```
0b00000001011111010010110000100000, L164. ADD $R11 $R26 $R11 \\ End of
     Algorithm 4.17
0b10101101011100100000000000000000, L165. SW $R11 $R18 0
0b00000001000001000100000000100000, L166. ADD $R8 $R4 $R8
0b10101101000100110000000000000000, L167. SW $R8 $R19 0
0b00000001000011000101110000100010, L168. SUB $R8 $R12 $R11 \\ Start of
     Algorithm 4.17
0b00000001011001100101110000111000, L169. DWTA $R11 $R6 $R11
0b00000001011111010010110000100000, L170. ADD $R11 $R26 $R11 \\ End of
     Algorithm 4.17
0b10101101011100110000000000000000, L171. SW $R11 $R19 0
0b00000001000001000100000000100000, L172. ADD $R8 $R4 $R8 \\ End of
     Algorithm 4.16
0b00000001001000100100100000100000, L173. ADD $R9 $R2 $R9
0b00010000000000000000000001100101, L174. BEQ $R0 $R0 101 \\ end while for
     $R9 < $R3 do
0b00000000000000000000000000000000, L175. SLL $R0 $R0 $R0
0b00000000000000000100100000100000, L176. ADD $R0 $R0 $R9
0b00000001010000010101000000100000, L177. ADD $R10 $R1 $R10
0b00000001010000000011100000100000, L178. ADD $R10 $R0 $R7
0b00000000111011000011100000100000, L179. ADD $R7 $R12 $R7
0b00000001010000000100000000100000, L180. ADD $R10 $R0 $R8
0b00000001000011000100000000100000, L181. ADD $R8 $R12 $R8
0b00010000000000000000000001100010, L182. BEQ $R0 $R0 98 \\ end while for
     $R10 < $R4 do
0b00000000000000000000000000000000, L183. SLL $R0 $R0 $R0
0b00000000011000000001100000000010, L184. SRL $R3 $R0 $R3 \\ Start of
     Algorithm 4.18
0b00000000100000000010000000000010, L185. SRL $R4 $R0 $R4
0b00000000101000000010100000000010, L186. SRL $R5 $R0 $R5
0b00000000101000000010100000000010, L187. SRL $R5 $R0 $R5
0b00000000110000010011000000100000, L188. ADD $R6 $R1 $R6
0b00100000000101100000000000000101, L189. ADDI $R0 $R22 5
0b00000000110101101011000000101010, L190. SLT $R6 $R22 $R22
0b00010110110000010000000011001101, L191. BNE $R22 $R1 205
0b00000000000000000000000000000000, L192. SLL $R0 $R0 $R0
0b00100000000110000000100000000000, L193. ADDI $R0 $R24 2048
0b00100000000110010001000000000000, L194. ADDI $R0 $R25 4096
0b00000000000000000110110000010000, L195. ADD $R0 $R0 $R27
0b10001111000101100000000000000000, L196. LW $R24 $R22 0
0b10101111001101100000000000000000, L197. SW $R25 $R22 0
0b00000011000000011100000000100000, L198. ADD $R24 $R1 $R24
0b00000011001000011100100000100000, L199. ADD $R25 $R1 $R25
0b00000011011000011101100000100000, L200. ADD $R27 $R1 $R27
0b00010111011001010000000011000100, L201. BNE $R27 $R5 196
0b00000000000000000000000000000000, L202. SLL $R0 $R0 $R0
0b00010000000000000000000000010010, L203. BEQ $R0 $R0 18
0b00000000000000000000000000000000, L204. SLL $R0 $R0 $R0
0b00000000000000000000000000000000, L205. NOP                  \\ End of
     Algorithm 4.18
    }
    with {block = 1};

    //************************************************************//
    // Signal Lines Declarations (requires no actual memory storage)
    //************************************************************//
    // Instruction Fetch (IF) Stage ------------------------------
    signal unsigned int 16          Signal_IF_PC;               //
        Intermediate signal: PC value
    signal unsigned int 16          Signal_IF_PC_Mux;           //
        Intermediate signal: Output from PC_Mux
    signal unsigned int 16          Signal_IF_PC_Increment;     //
        Intermediate signal: PC + 1
    signal unsigned int 32          Signal_IF_Inst;             //
        Intermediate signal: Instructions from INST_MEMORY
```

```
// Instruction Decode (ID) Stage ------------------------------
signal unsigned int 32          Signal_ID_Inst;             //
     Intermediate signal: Instruction read from IFID Pipeline
signal unsigned int 16          Signal_ID_ReadReg1;         //
     Intermediate signal: Read Data 1 out from Register (before
     forwarding)
signal unsigned int 16          Signal_ID_ReadReg2;         //
     Intermediate signal: Read Data 2 out from Register(before
     forwarding)
signal unsigned int 16          Signal_ID_ReadData1;        //
     Intermediate signal: Read Data 1 out from Register (after
     forwarding)
signal unsigned int 16          Signal_ID_ReadData2;        //
     Intermediate signal: Read Data 2 out from Register (after
     forwarding)
//signal unsigned int 32          Signal_ID_SignExt;          //
     Intermediate signal: Extended signal from 16 to 32 bits

// Branch Unit
signal unsigned int 1           Signal_ID_Comp;             //
     Intermediate signal: Comparator output 1 - equal, 0 - not equal
signal unsigned int 1           Signal_ID_XOROut;           //
     Intermediate signal: Output from Branch Unit into ID AND gate

// Operation Control Signals
signal unsigned int 1           Signal_ID_RegDst;           // Control
     signal: Controls Register Destination
signal unsigned int 1           Signal_ID_ALUSrc;           // Control
     signal: Controls Source into ALU
signal unsigned int 1           Signal_ID_ALUOp1;           // Control
     signal: Controls ALU Operation (bit 1)
signal unsigned int 1           Signal_ID_ALUOp0;           // Control
     signal: Controls ALU Operation (bit 0)
signal unsigned int 1           Signal_ID_Branch;           // Control
     signal: Generates branch signal on branch instructions
signal unsigned int 1           Signal_ID_MemRead;          // Control
     signal: Allows memory read when asserted
signal unsigned int 1           Signal_ID_MemWrite;         // Control
     signal: Allows memory write when asserted
signal unsigned int 1           Signal_ID_MemtoReg;         // Control
     signal: Allows memory read data to Register when asserted
signal unsigned int 1           Signal_ID_RegWrite;         // Control
     signal: Allows Register write when asserted

// ID Mux A and B Control Signals
signal unsigned int 1           Signal_ID_MuxA_Ctrl;        // Control
     signal: Forwarding from WB stage for ReadData1
signal unsigned int 1           Signal_ID_MuxB_Ctrl;        // Control
     signal: Forwarding from WB stage for ReadData2

// IF Stage Control Signals from ID stage
signal unsigned int 1           Signal_ID_PCMux_Ctrl;       // Control
     signal: IF PCMUX control
signal unsigned int 1           Signal_ID_PCWrite;          // Control
     signal: IF PCWrite control
signal unsigned int 1           Signal_ID_IFIDWrite;        // Control
     signal: Pipeline Register write control

// Hazard Detection Unit
signal unsigned int 1           Signal_ID_HazardMux_Ctrl;   // Control
     signal: Selects zero for all controls and for Rs and Rt into the ID
     /EXMEM pipeline

// Hazard Selection Outputs
signal unsigned int 1           Signal_ID_Hzd_RegDst;       // Control
     signal: Controls Register Destination
```

```
signal unsigned int 1           Signal_ID_Hzd_ALUSrc;       // Control
    signal: Controls Source into ALU
signal unsigned int 1           Signal_ID_Hzd_ALUOp1;       // Control
    signal: Controls ALU Operation (bit 1)
signal unsigned int 1           Signal_ID_Hzd_ALUOp0;       // Control
    signal: Controls ALU Operation (bit 0)
signal unsigned int 1           Signal_ID_Hzd_MemRead;      // Control
    signal: Allows memory read when asserted
signal unsigned int 1           Signal_ID_Hzd_MemWrite;     // Control
    signal: Allows memory write when asserted
signal unsigned int 1           Signal_ID_Hzd_MemtoReg;     // Control
    signal: Allows memory read data to Register when asserted
signal unsigned int 1           Signal_ID_Hzd_RegWrite;     // Control
    signal: Allows Register write when asserted

signal unsigned int 5           Signal_ID_Hzd_Rs;           //
    Intermediate signal: value from Rs field(R-format)/also for
    forwarding
signal unsigned int 5           Signal_ID_Hzd_Rt;           //
    Intermediate signal: value from Rt field(R-format)/Rd field(I-
    format)
signal unsigned int 5           Signal_ID_Hzd_Rd;           //
    Intermediate signal: value from Rd field(R-format)

// Instruction Execution and Memory (EXMEM) Stage ------------
signal unsigned int 16          Signal_EXMEM_ReadData1;     //
    Intermediate signal: Read Data 1 out from Register
signal unsigned int 16          Signal_EXMEM_ReadData2;     //
    Intermediate signal: Read Data 2 out from Register
signal unsigned int 16          Signal_EXMEM_ImmVal;        //
    Intermediate signal: Immediate value from Inst[15:0]
signal unsigned int 16          Signal_EXMEM_ALUIn2;        //
    Intermediate signal: ALU input 2
signal unsigned int 16          Signal_EXMEM_ALU_Result;    //
    Intermediate signal: ALU result
signal unsigned int 16          Signal_EXMEM_MemData;       //
    Intermediate signal: Data from DATAMEM
signal unsigned int 16          Signal_EXMEM_WBData;        //
    Intermediate signal: Writeback Data

signal unsigned int 5           Signal_EXMEM_Rs;            //
    Intermediate signal: value from Rs field(R-format)/also for
    forwarding
signal unsigned int 5           Signal_EXMEM_Rt;            //
    Intermediate signal: value from Rt field(R-format)/Rd field(I-
    format)
signal unsigned int 5           Signal_EXMEM_Rd;            //
    Intermediate signal: value from Rd field(R-format)
signal unsigned int 5           Signal_EXMEM_Regd;          //
    Intermediate signal: selected destination register

signal unsigned int 1           Signal_EXMEM_RegDst;        // Control
    signal: Controls Register Destination
signal unsigned int 1           Signal_EXMEM_ALUSrc;        // Control
    signal: Controls Source into ALU
signal unsigned int 1           Signal_EXMEM_ALUOp1;        // Control
    signal: Controls ALU Operation (bit 1)
signal unsigned int 1           Signal_EXMEM_ALUOp0;        // Control
    signal: Controls ALU Operation (bit 0)
signal unsigned int 1           Signal_EXMEM_MemRead;       // Control
    signal: Allows memory read when asserted
signal unsigned int 1           Signal_EXMEM_MemWrite;      // Control
    signal: Allows memory write when asserted
signal unsigned int 1           Signal_EXMEM_MemtoReg;      // Control
    signal: Allows memory read data to Register when asserted
signal unsigned int 1           Signal_EXMEM_RegWrite;      // Control
    signal: Allows Register write when asserted
```

```
signal unsigned int 16          Signal_EXMEM_FA_Mux;        //
    Intermediate signal: output from Forwarding MUX A, also ALUIn1
signal unsigned int 16          Signal_EXMEM_FB_Mux;        //
    Intermediate signal: output from Forwarding MUX B

signal unsigned int 1           Signal_EXMEM_FA;            // Control
    signal: Forwarding MUX A data selection
signal unsigned int 1           Signal_EXMEM_FB;            // Control
    signal: Forwarding MUX B data selection

signal unsigned int 4           Signal_EXMEM_ALU_Ctrl;      // Control
    signal: Determines specific ALU operation
signal unsigned int 1           Signal_EXMEM_ALU_Ctrl_B3;   // Control
    signal: Determines specific ALU operation (bit-3)
signal unsigned int 1           Signal_EXMEM_ALU_Ctrl_B2;   // Control
    signal: Determines specific ALU operation (bit-2)
signal unsigned int 1           Signal_EXMEM_ALU_Ctrl_B1;   // Control
    signal: Determines specific ALU operation (bit-1)
signal unsigned int 1           Signal_EXMEM_ALU_Ctrl_B0;   // Control
    signal: Determines specific ALU operation (bit-0)

// Write Back (WB) Stage
signal unsigned int 1           Signal_WB_MemtoReg;         // Control
    signal: Allows memory read data to Register when asserted
signal unsigned int 1           Signal_WB_RegWrite;         // Control
    signal: Allows Register write when asserted
signal unsigned int 5           Signal_WB_Rd;               //
    Intermediate signal: Register destination addr
signal unsigned int 16          Signal_WB_ALUData;          //
    Intermediate signal: ALU Result
signal unsigned int 16          Signal_WB_MemData;          //
    Intermediate signal: Memory Read Data
signal unsigned int 16          Signal_WB_Mem_Data;         //
    Intermediate signal: Memory out
signal unsigned int 16          Signal_WB_ALU_Result;       //
    Intermediate signal: ALU result
signal unsigned int 16          Signal_WB_Data;             //
    Intermediate signal: Write back data

//*********************************************************//
// Components Declarations (requires actual memory storage)
//*********************************************************//
// 32-bit General Purpose Registers
unsigned int 16         Register[32];                       // 32
    general purpose registers

// Misc Registers
unsigned int 1          RUN;                                // 1 - Run
    processor / 0 - Terminate
unsigned int 32         ClockCycle;                         // Clock
    cycle counter

// IF/ID Stage Registers
unsigned int 16         PC;                                 // Program
    Counter
unsigned int 32         IFID_Inst;                          // IF/ID
    Register: Holds instructions from INST_MEMORY

// ID/EXMEM Stage Registers
unsigned int 1          IDEXMEM_WB_MemtoReg;                // ID/EXMEM
    Register: Holds WB MemtoReg control signal
unsigned int 1          IDEXMEM_WB_RegWrite;                // ID/EXMEM
    Register: Holds WB RegWrite control signal
unsigned int 1          IDEXMEM_MEM_MemRead;                // ID/EXMEM
    Register: Holds MEM MemRead control signal
unsigned int 1          IDEXMEM_MEM_MemWrite;               // ID/EXMEM
    Register: Holds MEM MemWrite control signal
```

```
    unsigned int 1          IDEXMEM_EX_RegDst;                  // ID/EXMEM
         Register: Holds EX RegDst control signal
    unsigned int 1          IDEXMEM_EX_ALUOp1;                  // ID/EXMEM
         Register: Holds EX ALUOp1 control signal
    unsigned int 1          IDEXMEM_EX_ALUOp0;                  // ID/EXMEM
         Register: Holds EX ALUOp0 control signal
    unsigned int 1          IDEXMEM_EX_ALUSrc;                  // ID/EXMEM
         Register: Holds EX ALUSrc control signal

    unsigned int 16         IDEXMEM_ReadData1;                  // ID/EXMEM
         Register: Holds read data 1 from Register
    unsigned int 16         IDEXMEM_ReadData2;                  // ID/EXMEM
         Register: Holds read data 2 from Register
    unsigned int 16         IDEXMEM_ImmVal;                     // ID/EXMEM
         Register: Holds immediate value for Instuction[15:0]
    unsigned int 5          IDEXMEM_Rs;                         // ID/EXMEM
         Register: Holds value from Rs field(R-format)/also for forwarding
    unsigned int 5          IDEXMEM_Rt;                         // ID/EXMEM
         Register: Holds value from Rt field(R-format)/Rd field(I-format)
    unsigned int 5          IDEXMEM_Rd;                         // ID/EXMEM
         Register: Holds value from Rd field(R-format)

    // EXMEM/WB Stage Registers
    unsigned int 1          EXMEMWB_WB_RegWrite;                // EXMEM/WB
         Register: Holds WB RegWrite control signal
    unsigned int 1          EXMEMWB_WB_MemtoReg;                // EXMEM/WB
         Register: Holds WB MemtoReg control signal
    unsigned int 16         EXMEMWB_WB_ALUData;                 // EXMEM/WB
         Register: Holds ALU Data
    unsigned int 16         EXMEMWB_WB_MemData;                 // EXMEM/WB
         Register: Holds Memory Read Data
    unsigned int 5          EXMEMWB_Rd;                         // EXMEM/WB
         Register: Holds register destination addr

    // Components Initialization
    RUN             = 1;                // Set RUN flag to 1
    ClockCycle      = 0;                // ClockCycle counter starts at 0
    PC              = 0;                // PC starts at 0
    Register[0]     = 0;
    Register[1]     = 1;
    Register[2]     = 2;

    // Processor Core
    while (RUN == 1) // run processor
    {
        par // begin parallel execution
        {
            //-------------------------------------------------------------
            // Intermediate signal values
            //-------------------------------------------------------------

            // Stage 1: Instruction Fetch -------------------------------
            Signal_IF_PC            = PC;
            Signal_IF_PC_Increment  = Signal_IF_PC + 1;

            // PC source selection
            if(Signal_ID_PCMux_Ctrl == 0) // if no branch occurs
            {
                Signal_IF_PC_Mux    = Signal_IF_PC_Increment;
            }
            else // if branch occurs
            {
                Signal_IF_PC_Mux    = Signal_ID_Inst[15:0];
            }

            // Instruction fetching from INST_MEMORY
            Signal_IF_Inst         = INST_MEMORY[Signal_IF_PC[10:0]];
```

```
// Stage 2: Instruction Decode ------------------------------
Signal_ID_Inst          = IFID_Inst;
Signal_ID_ReadReg1      = Register[Signal_ID_Inst[25:21]];
Signal_ID_ReadReg2      = Register[Signal_ID_Inst[20:16]];

Signal_ID_RegDst        = (~Signal_ID_Inst[29]) & (~
     Signal_ID_Inst[26]);
Signal_ID_ALUSrc        = Signal_ID_Inst[29] | ((~Signal_ID_Inst
     [28]) & Signal_ID_Inst[26]);
Signal_ID_MemtoReg      = Signal_ID_Inst[26];
Signal_ID_RegWrite      = ((~Signal_ID_Inst[29]) & (~
     Signal_ID_Inst[28])) |
                            ((~Signal_ID_Inst[28]) & (~
                                 Signal_ID_Inst[26]));
Signal_ID_MemRead       = (~Signal_ID_Inst[29]) & (~
     Signal_ID_Inst[28]) & Signal_ID_Inst[26];
Signal_ID_MemWrite      = Signal_ID_Inst[29] & Signal_ID_Inst
     [26];
Signal_ID_ALUOp1        = (~Signal_ID_Inst[29]) & (~
     Signal_ID_Inst[28]) & (~Signal_ID_Inst[26]);
Signal_ID_ALUOp0        = Signal_ID_Inst[28];
Signal_ID_Branch        = Signal_ID_Inst[28];

// Branch Unit consist of equality comparator and a XOR gate
// comparator logic for reducing branch hazard
// still requires one NOP instructions after branch instruction
     (conventional is 3 NOPs)
if(Signal_ID_ReadData1[15:0] == Signal_ID_ReadData2[15:0]) // if
      both data values are equal
{Signal_ID_Comp = 1;}
else{Signal_ID_Comp = 0;}

// Generate Branch Unit output
Signal_ID_XOROut        = Signal_ID_Comp ^ Signal_ID_Inst[26];

 // Generate MUX signal for selection of new PC value
Signal_ID_PCMux_Ctrl    = Signal_ID_Branch & Signal_ID_XOROut;

// Forwarding logic
if((Signal_WB_RegWrite == 1) && (Signal_WB_Rd != 0) && (
     Signal_WB_Rd == Signal_ID_Inst[25:21]))
{Signal_ID_MuxA_Ctrl = 1;}
else{Signal_ID_MuxA_Ctrl = 0;}

if((Signal_WB_RegWrite == 1) && (Signal_WB_Rd != 0) && (
     Signal_WB_Rd == Signal_ID_Inst[20:16]))
{Signal_ID_MuxB_Ctrl = 1;}
else{Signal_ID_MuxB_Ctrl = 0;}

// Forwarding ID Mux A data selection
if(Signal_ID_MuxA_Ctrl == 1)
{Signal_ID_ReadData1 = Signal_WB_Data;}
else{Signal_ID_ReadData1 = Signal_ID_ReadReg1;}

// Forwarding ID Mux B data selection
if(Signal_ID_MuxB_Ctrl == 1)
{Signal_ID_ReadData2 = Signal_WB_Data;}
else{Signal_ID_ReadData2 = Signal_ID_ReadReg2;}

// Register write
if(Signal_WB_RegWrite == 1)
{Register[Signal_WB_Rd] = Signal_WB_Data;}
else
{delay;}
```

```
            // Hazard Detection Unit
            if((Signal_ID_Branch == 1) && (((Signal_ID_Inst[25:21] ==
                 Signal_EXMEM_Rs) && (Signal_ID_Inst[25:21] != 0) && (
                 Signal_ID_Inst[25:21] != 1))
                || ((Signal_ID_Inst[25:21] == Signal_EXMEM_Regd) && (
                     Signal_ID_Inst[25:21] != 0) && (Signal_ID_Inst[25:21]
                     != 1))
                || ((Signal_ID_Inst[20:16] == Signal_EXMEM_Rs) && (
                     Signal_ID_Inst[20:16] != 0) && (Signal_ID_Inst[20:16]
                     != 1))
                || ((Signal_ID_Inst[20:16] == Signal_EXMEM_Regd) && (
                     Signal_ID_Inst[20:16] != 0) && (Signal_ID_Inst[20:16]
                     != 1))))
            {
                par
                {
                    Signal_ID_PCWrite        = 0;
                    Signal_ID_IFIDWrite      = 0;
                    Signal_ID_HazardMux_Ctrl = 1;
                }
            }
            else // no hazard
            {
                par
                {
                    Signal_ID_PCWrite        = 1;
                    Signal_ID_IFIDWrite      = 1;
                    Signal_ID_HazardMux_Ctrl = 0;
                }
            }

            if(Signal_ID_HazardMux_Ctrl == 0)
            {
                par
                {
                    Signal_ID_Hzd_Rs         = Signal_ID_Inst[25:21];
                    Signal_ID_Hzd_Rt         = Signal_ID_Inst[20:16];
                    Signal_ID_Hzd_Rd         = Signal_ID_Inst[15:11];
                }
            }
            else
            {
                par
                {
                    Signal_ID_Hzd_Rs         = 0;
                    Signal_ID_Hzd_Rt         = 0;
                    Signal_ID_Hzd_Rd         = 0;
                }
            }

            if(Signal_ID_HazardMux_Ctrl == 0)
            {
                par
                {
                    Signal_ID_Hzd_MemtoReg   = Signal_ID_MemtoReg;
                    Signal_ID_Hzd_RegWrite   = Signal_ID_RegWrite;
                    Signal_ID_Hzd_MemRead    = Signal_ID_MemRead;
                    Signal_ID_Hzd_MemWrite   = Signal_ID_MemWrite;
                    Signal_ID_Hzd_RegDst     = Signal_ID_RegDst;
                    Signal_ID_Hzd_ALUOp1     = Signal_ID_ALUOp1;
                    Signal_ID_Hzd_ALUOp0     = Signal_ID_ALUOp0;
                    Signal_ID_Hzd_ALUSrc     = Signal_ID_ALUSrc;
                }
            }
            else
            {
                par
```

```
    {
        Signal_ID_Hzd_MemtoReg   = 0;
        Signal_ID_Hzd_RegWrite   = 0;
        Signal_ID_Hzd_MemRead    = 0;
        Signal_ID_Hzd_MemWrite   = 0;
        Signal_ID_Hzd_RegDst     = 0;
        Signal_ID_Hzd_ALUOp1     = 0;
        Signal_ID_Hzd_ALUOp0     = 0;
        Signal_ID_Hzd_ALUSrc     = 0;
    }
}

// Stage 3: Execution ----------------------------------------
Signal_EXMEM_ReadData1     = IDEXMEM_ReadData1;
Signal_EXMEM_ReadData2     = IDEXMEM_ReadData2;
Signal_EXMEM_ImmVal        = IDEXMEM_ImmVal;

Signal_EXMEM_Rs            = IDEXMEM_Rs;
Signal_EXMEM_Rt            = IDEXMEM_Rt;
Signal_EXMEM_Rd            = IDEXMEM_Rd;

Signal_EXMEM_MemtoReg      = IDEXMEM_WB_MemtoReg;
Signal_EXMEM_RegWrite      = IDEXMEM_WB_RegWrite;
Signal_EXMEM_MemRead       = IDEXMEM_MEM_MemRead;
Signal_EXMEM_MemWrite      = IDEXMEM_MEM_MemWrite;
Signal_EXMEM_RegDst        = IDEXMEM_EX_RegDst;
Signal_EXMEM_ALUOp1        = IDEXMEM_EX_ALUOp1;
Signal_EXMEM_ALUOp0        = IDEXMEM_EX_ALUOp0;
Signal_EXMEM_ALUSrc        = IDEXMEM_EX_ALUSrc;

// Data forwarding logic
if((Signal_WB_RegWrite == 1) && (Signal_WB_Rd != 0) && (
     Signal_WB_Rd == Signal_EXMEM_Rs))
{Signal_EXMEM_FA = 1;}
else
{Signal_EXMEM_FA = 0;}

if((Signal_WB_RegWrite == 1) && (Signal_WB_Rd != 0) && (
     Signal_WB_Rd == Signal_EXMEM_Rt))
{Signal_EXMEM_FB = 1;}
else
{Signal_EXMEM_FB = 0;}

// Forwarding A MUX data selection
if(Signal_EXMEM_FA == 0)        // $R1 data is available (no
     dependency)
{Signal_EXMEM_FA_Mux = Signal_EXMEM_ReadData1;}
else
{Signal_EXMEM_FA_Mux = Signal_WB_Data;}
{delay;}

// Forwarding B MUX data selection
if(Signal_EXMEM_FB == 0)        // $R2 data is available (no
     dependency)
{Signal_EXMEM_FB_Mux = Signal_EXMEM_ReadData2;}
else
{Signal_EXMEM_FB_Mux = Signal_WB_Data;}
{delay;}

// ALU Input 2 data selection
if(Signal_EXMEM_ALUSrc == 0)               // R-FORMAT, input
     from rt field
{Signal_EXMEM_ALUIn2 = Signal_EXMEM_FB_Mux;}
else                                       // I-FORMAT, input
     from const/addr field
{Signal_EXMEM_ALUIn2 = Signal_EXMEM_ImmVal;}
```

```
// ALU Control
Signal_EXMEM_ALU_Ctrl_B3   = (Signal_EXMEM_ALUOp1 &
     Signal_EXMEM_ImmVal[4]) | (Signal_EXMEM_ALUOp1 &
                                (~Signal_EXMEM_ImmVal[5]) &
                                     Signal_EXMEM_ImmVal[2]);
Signal_EXMEM_ALU_Ctrl_B2   = (Signal_EXMEM_ALUOp1 &
     Signal_EXMEM_ImmVal[3]) | (Signal_EXMEM_ALUOp1 &
                                (~Signal_EXMEM_ImmVal[5]) & (~
                                     Signal_EXMEM_ImmVal[2])) |
                                      (Signal_EXMEM_ALUOp1 &
                                Signal_EXMEM_ImmVal[2] &
                                     Signal_EXMEM_ImmVal[1]);
Signal_EXMEM_ALU_Ctrl_B1   = (Signal_EXMEM_ALUOp1 &
     Signal_EXMEM_ImmVal[4] & (~Signal_EXMEM_ImmVal[3])) |
                                (Signal_EXMEM_ALUOp1 & (~
                                     Signal_EXMEM_ImmVal[5]) &
                                     Signal_EXMEM_ImmVal[1]) |
                                (Signal_EXMEM_ALUOp1 &
                                     Signal_EXMEM_ImmVal[5] &
                                     Signal_EXMEM_ImmVal[2] &
                                (~Signal_EXMEM_ImmVal[1])) | (
                                     Signal_EXMEM_ALUOp1 &
                                     Signal_EXMEM_ImmVal[3] &
                                Signal_EXMEM_ImmVal[1]) | (
                                     Signal_EXMEM_ALUOp1 & (~
                                     Signal_EXMEM_ImmVal[3]) &
                                (~Signal_EXMEM_ImmVal[2]) &
                                     Signal_EXMEM_ImmVal[0]);
Signal_EXMEM_ALU_Ctrl_B0   = (Signal_EXMEM_ALUOp1 &
     Signal_EXMEM_ImmVal[0]) | (Signal_EXMEM_ALUOp1 &
                                (~Signal_EXMEM_ImmVal[5]) & (~
                                     Signal_EXMEM_ImmVal[2]) &
                                (~Signal_EXMEM_ImmVal[1])) | (
                                     Signal_EXMEM_ALUOp1 &
                                     Signal_EXMEM_ImmVal[5] &
                                (~Signal_EXMEM_ImmVal[4]) & (~
                                     Signal_EXMEM_ImmVal[2]) &
                                Signal_EXMEM_ImmVal[1]) | (
                                     Signal_EXMEM_ALUOp1 &
                                     Signal_EXMEM_ImmVal[5] &
                                (~Signal_EXMEM_ImmVal[3]) & (~
                                     Signal_EXMEM_ImmVal[2]) &
                                     Signal_EXMEM_ImmVal[1]);
Signal_EXMEM_ALU_Ctrl      = Signal_EXMEM_ALU_Ctrl_B3 @
     Signal_EXMEM_ALU_Ctrl_B2
                                @ Signal_EXMEM_ALU_Ctrl_B1 @
                                     Signal_EXMEM_ALU_Ctrl_B0;

// ALU
if(Signal_EXMEM_ALU_Ctrl == 0b0000)       // ADD
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
         Signal_EXMEM_FA_Mux[15:0]) + ((signed)
         Signal_EXMEM_ALUIn2[15:0])));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0001)  // SUB
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
         Signal_EXMEM_FA_Mux[15:0]) - ((signed)
         Signal_EXMEM_ALUIn2[15:0])));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0010)  // AND
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
         Signal_EXMEM_FA_Mux[15:0]) & ((signed)
         Signal_EXMEM_ALUIn2[15:0])));
}
```

```
else if(Signal_EXMEM_ALU_Ctrl == 0b0011)  // OR
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
        Signal_EXMEM_FA_Mux[15:0]) | ((signed)
        Signal_EXMEM_ALUIn2[15:0])));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0100)  // NOT
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(~((signed)
        Signal_EXMEM_FA_Mux[15:0])));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0101)  // SLL
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
        Signal_EXMEM_FA_Mux[15:0]) << 1));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0110)  // SRL
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
        Signal_EXMEM_FA_Mux[15:0]) >> 1));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b0111)  // SLT
{
    if(((signed)Signal_EXMEM_FA_Mux[15:0]) < ((signed)
        Signal_EXMEM_ALUIn2[15:0]))
    {Signal_EXMEM_ALU_Result = 1;}
    else
    {Signal_EXMEM_ALU_Result = 0;}
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1000)  // ABS
{
    // To perform absolute, invert all bits is bit 12 is 1, and
        then add 1 to the result
    if(Signal_EXMEM_FA_Mux[15] == 1)
    {
        Signal_EXMEM_ALU_Result = 0@((~Signal_EXMEM_FA_Mux
            [15:0]) + 1);
    }
    else
    {
        Signal_EXMEM_ALU_Result = 0@(Signal_EXMEM_FA_Mux[15:0]);
    }
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1001)  // ABSSUB
{
    // if second field is larger than the first field there will
         be a negative result before absolute
    if(((signed)Signal_EXMEM_FA_Mux[15:0]) < ((signed)
        Signal_EXMEM_ALUIn2[15:0]))
    {
        Signal_EXMEM_ALU_Result = 0@((~((unsigned)(((signed)
            Signal_EXMEM_FA_Mux[15:0]) - ((signed)
            Signal_EXMEM_ALUIn2[15:0])))) + 1);
    }
    else // normal subtraction
    {
        Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
            Signal_EXMEM_FA_Mux[15:0]) - ((signed)
            Signal_EXMEM_ALUIn2[15:0])));
    }
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1010)  // ADDS
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
        Signal_EXMEM_FA_Mux[15:0] + (signed)Signal_EXMEM_ALUIn2
        [15:0]) >> 1));
}
```

```
else if(Signal_EXMEM_ALU_Ctrl == 0b1011)  // ADDSS
{
    Signal_EXMEM_ALU_Result = 0@((unsigned)(((signed)
         Signal_EXMEM_FA_Mux[15:0] + (signed)Signal_EXMEM_ALUIn2
         [15:0] + (signed)2) >> 2));
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1100)  // DWT ADDRESS
     REARRANGE
{
    if(Signal_EXMEM_ALUIn2[14:0] == 1) // Lv1
    {
        Signal_EXMEM_ALU_Result = 0[15:11] @ Signal_EXMEM_FA_Mux
             [7] @ Signal_EXMEM_FA_Mux[0] @
                                    Signal_EXMEM_FA_Mux[10:8] @
                                         Signal_EXMEM_FA_Mux
                                         [6:1];
    }
    else if(Signal_EXMEM_ALUIn2[14:0] == 2) // Lv3
    {
        Signal_EXMEM_ALU_Result = 0[15:9] @ Signal_EXMEM_FA_Mux
             [6] @ Signal_EXMEM_FA_Mux[0] @
                                    Signal_EXMEM_FA_Mux[8:7] @
                                         Signal_EXMEM_FA_Mux
                                         [5:1];
    }
    else if(Signal_EXMEM_ALUIn2[14:0] == 3) // Lv3
    {
        Signal_EXMEM_ALU_Result = 0[15:7] @ Signal_EXMEM_FA_Mux
             [5] @ Signal_EXMEM_FA_Mux[0] @
                                    Signal_EXMEM_FA_Mux[6] @
                                         Signal_EXMEM_FA_Mux
                                         [4:1];
    }
    else if(Signal_EXMEM_ALUIn2[14:0] == 4) // Lv4
    {
        Signal_EXMEM_ALU_Result = 0[15:5] @ Signal_EXMEM_FA_Mux
             [4] @ Signal_EXMEM_FA_Mux[0] @
                                    Signal_EXMEM_FA_Mux[3:1];
    }
    else
    {delay;}
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1101)  // FREE
{
    delay;
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1110)  // FREE
{
    delay;
}
else if(Signal_EXMEM_ALU_Ctrl == 0b1111)  // FREE
{
    delay;
}
else
{delay;}

// Destination register selection from either rd(R-FORMAT) or rt
     (I-FORMAT)
if(Signal_EXMEM_RegDst == 1)
{Signal_EXMEM_Regd = Signal_EXMEM_Rd;}
else
{Signal_EXMEM_Regd = Signal_EXMEM_Rt;}

// Memory Read / Write
if((Signal_EXMEM_MemRead == 1) && (Signal_EXMEM_MemWrite != 1))
```

```
{
    Signal_EXMEM_MemData = 0@STRIP_MEMORY[Signal_EXMEM_FA_Mux
        [12:0]];
}
else
{delay;}

if((Signal_EXMEM_MemRead != 1) && (Signal_EXMEM_MemWrite == 1))
{
    STRIP_MEMORY[Signal_EXMEM_FA_Mux[12:0]] =
        Signal_EXMEM_FB_Mux[15:0];
}
else
{delay;}

// Stage 4: Write Back ----------------------------------------
Signal_WB_RegWrite          = EXMEMWB_WB_RegWrite;
Signal_WB_Rd                = EXMEMWB_Rd;
Signal_WB_MemtoReg          = EXMEMWB_WB_MemtoReg;
Signal_WB_ALUData           = EXMEMWB_WB_ALUData;
Signal_WB_MemData           = EXMEMWB_WB_MemData;

if(Signal_WB_MemtoReg == 1)
{Signal_WB_Data = Signal_WB_MemData;}
else
{Signal_WB_Data = Signal_WB_ALUData;}

//-------------------------------------------------------------
// New clock cycle
//-------------------------------------------------------------

// Stage 1: Instruction Fetch ---------------------------------
if(Signal_ID_PCWrite == 1)
{PC = Signal_IF_PC_Mux;}
else{delay;}

if(Signal_ID_IFIDWrite == 1)
{IFID_Inst       = Signal_IF_Inst;}
else{delay;}

// Stage 2: Instruction Decode --------------------------------
IDEXMEM_WB_MemtoReg    = Signal_ID_Hzd_MemtoReg;
IDEXMEM_WB_RegWrite    = Signal_ID_Hzd_RegWrite;
IDEXMEM_MEM_MemRead    = Signal_ID_Hzd_MemRead;
IDEXMEM_MEM_MemWrite   = Signal_ID_Hzd_MemWrite;
IDEXMEM_EX_RegDst      = Signal_ID_Hzd_RegDst;
IDEXMEM_EX_ALUOp1      = Signal_ID_Hzd_ALUOp1;
IDEXMEM_EX_ALUOp0      = Signal_ID_Hzd_ALUOp0;
IDEXMEM_EX_ALUSrc      = Signal_ID_Hzd_ALUSrc;

IDEXMEM_ReadData1      = Signal_ID_ReadData1;
IDEXMEM_ReadData2      = Signal_ID_ReadData2;
IDEXMEM_ImmVal         = Signal_ID_Inst[15:0];
IDEXMEM_Rs             = Signal_ID_Hzd_Rs;
IDEXMEM_Rt             = Signal_ID_Hzd_Rt;
IDEXMEM_Rd             = Signal_ID_Hzd_Rd;

// Stage 3: Execution and Memory Access -----------------------
EXMEMWB_WB_RegWrite       = Signal_EXMEM_RegWrite;
EXMEMWB_WB_MemtoReg       = Signal_EXMEM_MemtoReg;
EXMEMWB_Rd                = Signal_EXMEM_Regd;
EXMEMWB_WB_ALUData        = Signal_EXMEM_ALU_Result;
EXMEMWB_WB_MemData        = Signal_EXMEM_MemData;
```

```
            //-------------------------------------------------------------
            // Update counter / termination condition
            //-------------------------------------------------------------
            ClockCycle = ClockCycle + 1;    // increment clock cycle
            if(PC == InstMemSize)           // if reached the last
                 instruction
            {
                RUN = 0;                    // set termination condition
            }
            else {delay;}
        } // end par
    } // end while
}

static macro proc OutputStripMemory()
{
    unsigned int PatchCountBits      PatchCounter;
    unsigned int PatchPixelCountBits PatchPixelCounter, RCounter;
    unsigned int PatchWidthCountBits PatchWidthCounter;
    unsigned int StripBits           WriteData;
    unsigned int RAMStripAddressBits StartStripAddressPatchData;
    unsigned int RAMStripAddressBits StartStripAddressPatchHeader;
    unsigned int PatchCountBits      RowPatchCounter;
    unsigned int PatchCountBits      ColPatchCounter;
    unsigned int PatchCountBits      RAMAddressCounter;

    // Zigbee transmission variables - using API packet
    unsigned 8 StartDelimiter, FrameType, FrameID, BroadcastRadius, Options;
    unsigned 8 DestAddr64_0, DestAddr64_1, DestAddr64_2, DestAddr64_3,
         DestAddr64_4, DestAddr64_5, DestAddr64_6, DestAddr64_7;
    unsigned 8 DestAddr16_MSB, DestAddr16_LSB;
    unsigned 16 Length, NumofPatches, TempCheck, CheckSum, PatchWidthNum;

    seq
    {
        // Parameter settings which cannot be assigned to unsigned from
             macro expr goes here
        NumofPatches = 32; // number of patches used per strip image
        PatchWidthNum = 8;

        // First send data containing valid patches information (saliency
             bit = 1 or 0)
        StartDelimiter = 0x7E;
        Length = 14 + NumofPatches;
        FrameType = 0x10;
        FrameID = 0x01;
        DestAddr64_0 = 0x00;
        DestAddr64_1 = 0x00;
        DestAddr64_2 = 0x00;
        DestAddr64_3 = 0x00;
        DestAddr64_4 = 0x00;
        DestAddr64_5 = 0x00;
        DestAddr64_6 = 0x00;
        DestAddr64_7 = 0x00;
        DestAddr16_MSB = 0xFF;
        DestAddr16_LSB = 0xFE;
        BroadcastRadius = 0x00;
        Options = 0x00;

        TempCheck = 0;
        TempCheck = 0@(FrameType + FrameID + DestAddr64_0 + DestAddr64_1 +
             DestAddr64_2 + DestAddr64_3 + DestAddr64_4 + DestAddr64_5 +
             DestAddr64_6 + DestAddr64_7 + DestAddr16_MSB + DestAddr16_LSB +
              BroadcastRadius + Options);

        // Send API packet header
        RC10RS232Write(StartDelimiter);
```

```
RC10RS232Write(Length[15:8]);
RC10RS232Write(Length[7:0]);
RC10RS232Write(FrameType);
RC10RS232Write(FrameID);
RC10RS232Write(DestAddr64_0);
RC10RS232Write(DestAddr64_1);
RC10RS232Write(DestAddr64_2);
RC10RS232Write(DestAddr64_3);
RC10RS232Write(DestAddr64_4);
RC10RS232Write(DestAddr64_5);
RC10RS232Write(DestAddr64_6);
RC10RS232Write(DestAddr64_7);
RC10RS232Write(DestAddr16_MSB);
RC10RS232Write(DestAddr16_LSB);
RC10RS232Write(BroadcastRadius);
RC10RS232Write(Options);

// Send API Data
RAMAddressStripMemory = RAMStripAddress - NumPatches;
PatchCounter = 0;
while (PatchCounter != NumPatches)
{
    seq
    {
        WriteData = STRIP_MEMORY[RAMAddressStripMemory];
        RC10RS232Write(WriteData[7:0]);
        TempCheck = TempCheck + (0@(WriteData[7:0]));
        RAMAddressStripMemory++;
        PatchCounter++;
    }
}

// Send API checksum
CheckSum = 0xFF - TempCheck;
RC10RS232Write(CheckSum[7:0]);

// Next transmit image data

StartStripAddressPatchHeader = RAMStripAddress - NumPatches;
StartStripAddressPatchData   = 0;
RowPatchCounter = 0;
ColPatchCounter = 0;

while(RowPatchCounter != PatchesPerRow)
{
    seq
    {
        while(ColPatchCounter != PatchesPerCol)
        {
            seq
            {
                RAMAddressStripMemory = StartStripAddressPatchHeader
                    ;
                if(STRIP_MEMORY[RAMAddressStripMemory] == 1)
                {
                    seq
                    {
                        RAMAddressStripMemory =
                            StartStripAddressPatchData;
                        PatchWidthCounter = 0;
                        PatchPixelCounter = 0;

                        StartDelimiter = 0x7E;
                        Length = 14 + 64;
                        FrameType = 0x10;
                        FrameID = 0x01;
                        DestAddr64_0 = 0x00;
```

```
DestAddr64_1 = 0x00;
DestAddr64_2 = 0x00;
DestAddr64_3 = 0x00;
DestAddr64_4 = 0x00;
DestAddr64_5 = 0x00;
DestAddr64_6 = 0x00;
DestAddr64_7 = 0x00;
DestAddr16_MSB = 0xFF;
DestAddr16_LSB = 0xFE;
BroadcastRadius = 0x00;
Options = 0x00;

TempCheck = 0;
TempCheck = 0@(FrameType + FrameID +
     DestAddr64_0 + DestAddr64_1 +
     DestAddr64_2 + DestAddr64_3 +
     DestAddr64_4 + DestAddr64_5 +
     DestAddr64_6 + DestAddr64_7 +
     DestAddr16_MSB + DestAddr16_LSB +
     BroadcastRadius + Options);

// Send API packet header
RC10RS232Write(StartDelimiter);
RC10RS232Write(Length[15:8]);
RC10RS232Write(Length[7:0]);
RC10RS232Write(FrameType);
RC10RS232Write(FrameID);
RC10RS232Write(DestAddr64_0);
RC10RS232Write(DestAddr64_1);
RC10RS232Write(DestAddr64_2);
RC10RS232Write(DestAddr64_3);
RC10RS232Write(DestAddr64_4);
RC10RS232Write(DestAddr64_5);
RC10RS232Write(DestAddr64_6);
RC10RS232Write(DestAddr64_7);
RC10RS232Write(DestAddr16_MSB);
RC10RS232Write(DestAddr16_LSB);
RC10RS232Write(BroadcastRadius);
RC10RS232Write(Options);

while(PatchPixelCounter != (PatchWidth*
     PatchHeight))
{
    seq
    {
        WriteData = STRIP_MEMORY[
             RAMAddressStripMemory];
        RC10RS232Write(WriteData[7:0]);
        TempCheck = TempCheck + (0@(
             WriteData[7:0]));
        if(PatchWidthCounter == (PatchWidth
             - 1))
        {
            par
            {
                PatchWidthCounter     = 0;
                RAMAddressStripMemory =
                     RAMAddressStripMemory +
                      (ImageWidth -
                     PatchWidth + 1);
            }
        }
        else
        {
            par
            {
                PatchWidthCounter++;
```

```
                                                RAMAddressStripMemory++;
                                            }
                                        }

                                        PatchPixelCounter++;
                                    }
                                }
                                // Send API checksum
                                CheckSum = 0xFF - TempCheck;
                                RC10RS232Write(CheckSum[7:0]);
                            }
                        }

                        StartStripAddressPatchHeader++;
                        StartStripAddressPatchData =
                            StartStripAddressPatchData + PatchWidth;
                        ColPatchCounter++;
                    }
                }
                RAMAddressCounter = 0;
                while(RAMAddressCounter != PatchHeight - 1)
                {
                    seq
                    {
                        StartStripAddressPatchData =
                            StartStripAddressPatchData + ImageWidth;
                        RAMAddressCounter++;
                    }
                }
                RowPatchCounter++;
                ColPatchCounter = 0;
            }
        }

    }
}
```

Listing A.1 Handel-C Program Listing for WMSN FPGA Processor

References

1. Achanta, R., Estrada, F., Wils, P., Süsstrunk, S.: Salient region detection and segmentation. In: Proceedings of the 6th International Conference on Computer Vision Systems (ICVS'08), pp. 66–75. Springer, Berlin (2008). URL http://dl.acm.org/citation.cfm?id=1788524.1788532
2. Achanta, R., Hemami, S., Estrada, F., Susstrunk, S.: Frequency-tuned salient region detection. In: IEEE Conference on Computer Vision and Pattern Recognition, 2009 (CVPR 2009), pp. 1597–1604 (2009). doi:10.1109/CVPR.2009.5206596
3. Acharya, T., Tsai, P.S.: JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures. Wiley-Interscience, Hoboken (2004)
4. Aghajan, H., Cavallaro, A.: Multi-Camera Networks: Principles and Applications. Academic, London (2009)
5. Akyildiz, I.F., Melodia, T., Chowdhury, K.R.: A survey on wireless multimedia sensor networks. Comput. Netw. **51**(4), 921–960 (2007). doi:10.1016/j.comnet.2006.10.002. URL http://dx.doi.org/10.1016/j.comnet.2006.10.002
6. Almalkawi, I., Zapata, M., Al-Karaki, J., Morillo-Pozo, J.: Wireless multimedia sensor networks: Current trends and future directions. Sensors (Basel) **10**(7), 6662–6717 (2010)
7. Altera: Altera Stratix-V. http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp (2013)
8. Altera: Nios II Processor. http://www.altera.com.my/devices/processor/nios2/ni2-index.html (2011)
9. Altera: Stratix II datasheet. http://www.altera.com.my/literature/hb/stx2/stx2_sii51002.pdf (2007)
10. Ammari, A., Jemai, A.: Multiprocessor platform-based design for multimedia. IET Comput. Digit. Tech. **3**(1), 52 –61 (2009). doi:10.1049/iet-cdt:20070168
11. Angelopoulou, M.E., Masselos, K., Cheung, P.Y., Andreopoulos, Y.: Implementation and comparison of the 5/3 lifting 2d discrete wavelet transform computation schedules on fpgas. J. Signal Process. Syst. **51**(1), 3–21 (2008). doi:10.1007/s11265-007-0139-5. URL http://dx.doi.org/10.1007/s11265-007-0139-5
12. ARM Ltd.: ARM architectures. http://www.arm.com/products/processors/index.php (2011)
13. Arnold, M.G.: Verilog Digital Computer Design: Algorithms into Hardware. Prentice Hall, Upper Saddle River (1999)
14. Bhattar, R.K., Ramakrishnan, K., Dasgupta, K.: Strip based coding for large images using wavelets. Signal Process. Image Commun. **17**(6), 441–456 (2002). doi:10.1016/S0923-5965(02)00019-X. URL http://www.sciencedirect.com/science/article/pii/S092359650200019X
15. Brown, M., Lowe, D.G.: Automatic panoramic image stitching using invariant features. Int. J. Comput. Vis. **74**(1), 59–73 (2007). doi:10.1007/s11263-006-0002-3. URL http://dx.doi.org/10.1007/s11263-006-0002-3

DOI 10.1007/978-3-642-38203-1

16. Bruce, N.D.B., Tsotsos, J.K.: Saliency, attention, and visual search: An information theoretic approach. J. Vis. **9**(3), 1–24 (2009)
17. Burt, P.J., Adelson, E.H.: A multiresolution spline with application to image mosaics. ACM Trans. Graph. **2**(4), 217–236 (1983). doi:10.1145/245.247. URL http://doi.acm.org/10.1145/245.247
18. Cadambi, S., Weener, J., Goldstein, S.C., Schmit, H., Thomas, D.E.: Managing pipeline-reconfigurable fpgas. In: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA '98), pp. 55–64. ACM, New York (1998). doi:10.1145/275107.275120. URL http://doi.acm.org/10.1145/275107.275120
19. Cardoso, J.F.: High-order contrasts for independent component analysis. Neural Comput. **11**, 157–192 (1999). doi:http://dx.doi.org/10.1162/089976699300016863. URL http://dx.doi.org/10.1162/089976699300016863
20. Cassereau, P.M.: Wavelet-based image coding. In: Watson, A.B. (ed.) Digital Images and Human Vision, pp. 12–21. The MIT Press, Cambridge (1993)
21. Celoxica: Celoxica RC10. www.europractice.rl.ac.uk/vendors/agility_rc10.pdf (2005)
22. Celoxica: Celoxica RC230E. http://babbage.cs.qc.edu/courses/cs345/Manuals/RC200_203%20Manual.pdf (2005)
23. Celoxica: DK Design Suite. http://www.europractice.stfc.ac.uk/vendors/agility_dk4.pdf (2005)
24. Celoxica: DK4 Handel-C Language Reference Manual. http://babbage.cs.qc.edu/courses/cs345/Manuals/HandelC.pdf (2005)
25. Charfi, Y., Wakamiya, N., Murata, M.: Network-adaptive image and video transmission in camera-based wireless sensor networks. In: IEEE International Conference on Distributed Smart Cameras, pp. 336–343 (2007). URL http://dblp.uni-trier.de/db/conf/icdsc/icdsc2007.html#CharfiWM07
26. Cheung, S.C.S., Kamath, C.: Robust background subtraction with foreground validation for urban traffic video. EURASIP J. Appl. Signal Process. **14**, 2330–2340 (2005). doi:10.1155/ASP.2005.2330. URL http://dx.doi.org/10.1155/ASP.2005.2330
27. Chew, L.W., Ang, L.M., Seng, K.P.: New virtual spiht tree structures for very low memory strip-based image compression. IEEE Signal Process. Lett. **15**, 389 –392 (2008). doi:10.1109/LSP.2008.920515
28. Chew, L.W., Chia, W.C., Ang, L.M., Seng, K.P.: Very low-memory wavelet compression architecture using strip-based processing for implementation in wireless sensor networks. EURASIP J. Embed. Syst. **2009**, 9:1–9:1 (2009). doi:10.1155/2009/479281. URL http://dx.doi.org/10.1155/2009/479281
29. Chew, L.W., Chia, W.C., Ang, L.M., Seng, K.P.: Low-memory video compression architecture using strip-based processing for implementation in wireless multimedia sensor networks. Int. J. Sens. Netw. **11**(1), 33–47 (2012). doi:10.1504/IJSNET.2012.045033. URL http://dx.doi.org/10.1504/IJSNET.2012.045033
30. Chia, W.C., Chew, L.W., Ang, L.M., Seng, K.P.: Low memory image stitching and compression for wmsn using strip-based processing. IJSNet **11**(1), 22–32 (2012). URL http://dblp.uni-trier.de/db/journals/ijsnet/ijsnet11.html#ChiaCAS12
31. Choi, S., Scrofano, R., Prasanna, V.K., Jang, J.W.: Energy-efficient signal processing using FPGAs. In: Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (FPGA '03), pp. 225–234. ACM, New York (2003). doi:10.1145/611817.611850. URL http://doi.acm.org/10.1145/611817.611850
32. Chrysafis, C., Ortega, A.: Line-based, reduced memory, wavelet image compression. IEEE Trans. Image Process. **9**(3), 378–389 (2000). doi:10.1109/83.826776
33. Compton, K., Hauck, S.: Reconfigurable computing: A survey of systems and software. ACM Comput. Surv. **34**(2), 171–210 (2002). doi:10.1145/508352.508353. URL http://doi.acm.org/10.1145/508352.508353
34. Crossbow Technology: XBow SPB400–Stargate Gateway Datasheet. http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf (2006)

35. Crossbow Technology: XBow TELOSB Datasheet. http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf (2007)
36. Cucchiara, R.: Multimedia surveillance systems. In: Proceedings of the Third ACM International Workshop on Video Surveillance & Sensor Networks (VSSN '05), pp. 3–10. ACM, New York (2005). doi:10.1145/1099396.1099399. URL http://doi.acm.org/10.1145/1099396.1099399
37. Czarlinska, A., Kundur, D.: Reliable event-detection in wireless visual sensor networks through scalar collaboration and game-theoretic consideration. IEEE Trans. Multimed. **10**(5), 675–690 (2008). doi:10.1109/TMM.2008.922775
38. Daubechies, I., Sweldens, W.: Factoring wavelet transforms into lifting steps. J. Fourier Anal. Appl. **4**(3), 247–269 (1998)
39. Elazary, L., Itti, L.: A bayesian model for efficient visual search and recognition. Vis. Res. **50**(14), 1338–1352 (2010)
40. Estrin, G.: Organization of computer systems: The fixed plus variable structure computer. In: Papers Presented at the May 3–5, 1960, Western Joint IRE-AIEE-ACM Computer Conference (IRE-AIEE-ACM '60) (Western), pp. 33–40. ACM, New York (1960). doi:10.1145/1460361.1460365. URL http://doi.acm.org/10.1145/1460361.1460365
41. Faraji, I., Didehban, M., Zarandi, H.: Analysis of transient faults on a mips-based dual-core processor. In: International Conference on Availability, Reliability, and Security, 2010 (ARES '10), pp. 125–130 (2010). doi:10.1109/ARES.2010.30
42. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981). doi:10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692
43. Foulsham, T., Underwood, G.: What can saliency models predict about eye movements? spatial and sequential aspects of fixations during encoding and recognition. J. Vis. **8**(2) (2008). doi:10.1167/8.2.6. URL http://www.journalofvision.org/content/8/2/6.abstract
44. Frintrop, S.: Computational visual attention. In: Computer Analysis of Human Behavior, pp. 69–101. Springer, London (2011)
45. Frintrop, S., Jensfelt, P.: Attentional landmarks and active gaze control for visual slam. IEEE Trans. Robot. **24**(5), 1054–1065 (2008). doi:10.1109/TRO.2008.2004977
46. Fry, T., Hauck, S.: SPIHT image compression on FPGAs. IEEE Trans. Circuits Syst. Video Technol. **15**(9), 1138–1147 (2005). doi:10.1109/TCSVT.2005.852625
47. Garcia, V.: Keypoints Extraction. http://www.mathworks.com/matlabcentral/fileexchange/17894-keypoint-extraction (2007)
48. Gautham, P., Parthasarathy, R., Balasubramanian, K.: Low-power pipelined mips processor design. In: Proceedings of the 2009 12th International Symposium on Integrated Circuits (ISIC '09), pp. 462–465 (2009)
49. Girod, B., Aaron, A., Rane, S., Rebollo-Monedero, D.: Distributed video coding. Proc. IEEE **93**(1), 71 –83 (2005). doi:10.1109/JPROC.2004.839619
50. Gray, A., Lee, C., Arabshahi, P., Srinivasan, J.: Object-oriented reconfigurable processing for wireless networks. In: IEEE International Conference on Communications, 2002 (ICC 2002), vol. 1, pp. 497–501 (2002). doi:10.1109/ICC.2002.996903
51. Guo, C., Zhang, L.: A novel multiresolution spatiotemporal saliency detection model and its applications in image and video compression. IEEE Trans. Image Process. **19**(1), 185–198 (2010). doi:10.1109/TIP.2009.2030969
52. Guo, W., Xu, C., Ma, S., Xu, M.: Visual attention based small object segmentation in natual images. In: 2010 17th IEEE International Conference on Image Processing (ICIP), pp. 1565–1568 (2010). doi:10.1109/ICIP.2010.5649841
53. Guo, X., Lu, Y., Wu, F., Zhao, D., Gao, W.: Wyner–ziv-based multiview video coding. IEEE Trans. Circuits Syst. Video Technol. **18**(6), 713–724 (2008). doi:10.1109/TCSVT.2008.920970. URL http://dx.doi.org/10.1109/TCSVT.2008.920970
54. Gürses, E., Akan, Ö.: Multimedia communication in wireless sensor networks. Ann. Telecommun. **60**(7), 872–900 (2005)

55. Haering, N., da Vitoria Lobo, N.: Visual Event Detection. The International Series in Video Computing. Springer, New York (2001). URL http://www.springer.com/computer/image+processing/book/978-0-7923-7436-7
56. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of Fourth Alvey Vision Conference, pp. 147–151 (1988)
57. HART Communication Protocol and Foundation: WirelessHART Overview. http://www.hartcomm.org/protocol/wihart/wireless_overview.html (2010)
58. Hasan, M., Rahman, M., Hasan, M., Hasan, M., Ali, M.: An improved pipelined processor architecture eliminating branch and jump penalty. In: 2010 Second International Conference on Computer Engineering and Applications (ICCEA), vol. 1, pp. 621–625 (2010). doi: 10.1109/ICCEA.2010.126
59. Hauser, J., Wawrzynek, J.: Garp: A mips processor with a reconfigurable coprocessor. In: Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1997, pp. 12–21 (1997). doi:10.1109/FPGA.1997.624600
60. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. **35**(11), 93–104 (2000). doi: 10.1145/356989.356998. URL http://doi.acm.org/10.1145/356989.356998
61. Ho-Phuoc, T., Guyader, N., Gurin-Dugu, A.: A functional and statistical bottom-up saliency model to reveal the relative contributions of low-level visual guiding factors. Cogn. Comput. **2**, 344–359 (2010). doi:10.1007/s12559-010-9078-8. URL http://dx.doi.org/10.1007/s12559-010-9078-8
62. Hu, F., Kumar, S.: QoS considerations in wireless sensor networks for telemedicine. In: Proceedings of SPIE ITCOM Conference, Orlando, FL (2003)
63. Hunter, R.D.M., Johnson, T.T.: Introduction to VHDL. Springer, Berlin (1995). http://www.springer.com/engineering/circuits+%26+systems/book/978-0-412-73130-3
64. International, D.: XBee ZNet 2.5 Module. http://www.digi.com/support/productdetail?pid=3261 (2012). Accessed Nov 2012
65. International, D.: XBee/XBee-Pro RF Module Datasheet. http://www.digi.com/pdf/ds_xbeezbmodules.pdf (2011). Accessed Nov 2012
66. Itti, L., Koch, C.: A saliency-based search mechanism for overt and covert shifts of visual attention. Vis. Res. **40**, 1489–1506 (2000)
67. Itti, L., Koch, C., Niebur, E.: A model of saliency-based visual attention for rapid scene analysis. IEEE Trans. Pattern Anal. Mach. Intell. **20**(11), 1254–1259 (1998). doi:10.1109/34.730558
68. Itti, L., Rees, G., Tsotsos, J.: Models of bottom-up attention and saliency. In: Neurobiology of Attention, pp. 576–582. Elsevier, San Diego (2005)
69. James, W.: The Principles of Psychology. American Science Series: Advanced Course, vol. 1. H. Holt, New York (1918). URL http://books.google.com.my/books?id=lbtE-xb5U-oC
70. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next century challenges: Mobile networking for Smart Dust. In: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), pp. 271–278. ACM, New York (1999). doi:10.1145/313451.313558. URL http://doi.acm.org/10.1145/313451.313558
71. Koch, C., Ullman, S.: Shifts in selective visual attention: Towards the underlying neural circuitry. Hum. Neurobiol. **4**(4), 219–27 (1985)
72. Kunt, M., Ikonomopoulos, A., Kocher, M.: Second-generation image-coding techniques. Proc. IEEE **73**(4), 549–574 (1985). doi:10.1109/PROC.1985.13184
73. Kwok, T.T.O., Kwok, Y.K.: Computation and energy efficient image processing in wireless sensor networks based on reconfigurable computing. In: Proceedings of the 2006 International Conference Workshops on Parallel Processing (ICPPW '06), pp. 43–50. IEEE Computer Society, Washington, DC (2006). doi:10.1109/ICPPW.2006.30. URL http://dx.doi.org/10.1109/ICPPW.2006.30
74. Lee, S.H., Shin, J.K., Lee, M.: Non-uniform image compression using biologically motivated saliency map model. In: Proceedings of the 2004 Intelligent Sensors, Sensor

Networks and Information Processing Conference, 2004, pp. 525–530 (2004) doi: 10.1109/ISSNIP.2004.1417516
75. Li, L.J., Su, H., Xing, E.P., Fei-Fei, L.: Object bank: A high-level image representation for scene classification and semantic feature sparsification. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge (2010)
76. Li, X., Li, T.: Ecomips: An economic MIPS CPU design on FPGA. In: Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications, 2004, pp. 291–294 (2004). doi:10.1109/IWSOC.2004.1319896
77. Liang, J., DeMenthon, D., Doermann, D.: Note: Mosaicing of camera-captured document images. Comput. Vis. Image Underst. **113**(4), 572–579 (2009). doi: 10.1016/j.cviu.2008.12.004. URL http://dx.doi.org/10.1016/j.cviu.2008.12.004
78. Libelium Comunicaciones Distribuidas S.L.: Libelium Waspmote technical guide. http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf (2013)
79. Lindeberg, T.: Feature detection with automatic scale selection. Int. J. Comput. Vis. **30**(2), 79–116 (1998). doi:10.1023/A:1008045108935. URL http://dx.doi.org/10.1023/A:1008045108935
80. Liu, L., Yuan, F.G., Zhang, F.: Development of wireless smart sensor for structural health monitoring. In: Proceedings of the 2005 Smart Structures and Materials - Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems, vol. 5765, No. 20, pp. 176–186 (2005). doi:10.1117/12.600206. URL http://dx.doi.org/10.1117/12.600206
81. Liu, T., Yuan, Z., Sun, J., Wang, J., Zheng, N., Tang, X., Shum, H.Y.: Learning to detect a salient object. IEEE Trans. Pattern Anal. Mach. Intell. **33**(2), 353 –367 (2011). doi:10.1109/TPAMI.2010.70
82. Liu, Y., Xie, G., Chen, P., Chen, J., Li, Z.: Designing an asynchronous FPGA processor for low-power sensor networks. In: International Symposium on Signals, Circuits and Systems, 2009 (ISSCS 2009), pp. 1–6 (2009). doi:10.1109/ISSCS.2009.5206091
83. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004). doi:10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94
84. Lukai Cai, S.V., Gajski, D.D.: Technical report cecs-03-11. Technical Report, Center for Embedded Computer Systems, University of California, Irvine, California, 2003
85. Magli, E., Mancin, M., Merello, L.: Low-complexity video compression for wireless sensor networks. In: Proceedings of the 2003 International Conference on Multimedia and Expo - Vol. 3 (ICME '03), pp. 585–588. IEEE Computer Society, Washington, DC (2003). URL http://dl.acm.org/citation.cfm?id=1170746.1171773
86. Mangharam, R., Rowe, A., Rajkumar, R.: FireFly: A cross-layer platform for real-time embedded wireless networks. R. Time Syst. **37**(3), 183–231 (2007). doi: 10.1007/s11241-007-9028-z. URL http://dx.doi.org/10.1007/s11241-007-9028-z
87. Martin, D., Fowlkes, C., Tal, D., Malik, J.: The berkeley segmentation dataset. http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/ (2007)
88. Medeiros, H., Park, J., Kak, A.: Distributed object tracking using a cluster-based kalman filter in wireless camera networks. IEEE J. Sel. Top. Signal Process. **2**(4), 448–463 (2008). doi:10.1109/JSTSP.2008.2001310
89. Mendi, E., Milanova, M.: Image segmentation with active contours based on selective visual attention. In: Proceedings of the 3rd WSEAS International Symposium on Wavelets Theory and Applications in Applied Mathematics, Signal Processing & Modern Science (WAV'09), pp. 79–84. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2009). URL http://dl.acm.org/citation.cfm?id=1561963.1561976
90. (MERL), M.E.R.L.: Stereoscopic Video Sequences. ftp://ftp.merl.com/pub/avetro/mvc-testseq/orig-yuv/exit/ (2005)
91. Merrill, W., Sohrabi, K., Girod, L., Elson, J., Newberg, F.: Open standard development platforms for distributed sensor networks. In: SPIE Unattended Ground Sensor Technologies and Applications IV, pp. 327–337 (2002)

92. Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. Int. J. Comput. Vis. **60**(1), 63–86 (2004). doi:10.1023/B:VISI.0000027790.02288.f2. URL http://dx.doi.org/10.1023/B:VISI.0000027790.02288.f2
93. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1615–1630 (2005). doi:10.1109/TPAMI.2005.188. URL http://dx.doi.org/10.1109/TPAMI.2005.188
94. Mills, A., Dudek, G.: Image stitching with dynamic elements. Image Vis. Comput. **27**(10), 1593–1602 (2009). doi:10.1016/j.imavis.2009.03.004. URL http://dx.doi.org/10.1016/j.imavis.2009.03.004
95. MIPS Technologies: MIPS architectures. http://www.mips.com/products/architectures/mips32/ (2008)
96. Mulligan, G.: The 6lowpan architecture. In: Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets '07), pp. 78–82. ACM, New York (2007). doi:10.1145/1278972.1278992. URL http://doi.acm.org/10.1145/1278972.1278992
97. Ngau, C., Ang, L.M., Seng, K.: Low memory visual saliency architecture for data reduction in wireless sensor networks. IET Wirel. Sens. Syst. **2**(2), 115 –127 (2012). doi:10.1049/iet-wss.2011.0038
98. Omnivision: Omnivision OV16820. http://www.ovt.com/products/sensor.php?id=116 (2012)
99. Ouerhani, N., Bracamonte, J., Hugli, H., Ansorge, M., Pellandini, F.: Adaptive color image compression based on visual attention. In: Proceedings of the 11th International Conference on Image Analysis and Processing, 2001, pp. 416–421 (2001). doi:10.1109/ICIAP.2001.957045
100. Patterson, D.A., Hennessy, J.L.: Computer Organization and Design: The Hardware/Software Interface, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2007)
101. Pearlman, W.A., Islam, A., Nagaraj, N., Said, A.: Efficient, low-complexity image coding with a set-partitioning embedded block coder. IEEE Trans. Circuits Syst. Video Technol. **14**(11), 1219–1235 (2004). doi:10.1109/TCSVT.2004.835150. URL http://dx.doi.org/10.1109/TCSVT.2004.835150
102. Pham, D.M., Aziz, S.: FPGA-based image processor architecture for wireless multimedia sensor network. In: 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing (EUC), pp. 100–105 (2011). doi:10.1109/EUC.2011.38
103. Pradhan, S.S., Ramchandran, K.: Distributed source coding using syndromes (discus): Design and construction. IEEE Trans. Inf. Theory **49**(3), 626–643 (2006). doi:10.1109/TIT.2002.808103. URL http://dx.doi.org/10.1109/TIT.2002.808103
104. Puri, R., Ramchandran, K.: Prism: An uplink-friendly multimedia coding paradigm. In: Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003 (ICASSP '03), vol. 4, pp. IV–856–859 (2003). doi:10.1109/ICASSP.2003.1202778
105. Quattoni, A., Torralba, A.: Indoor scene recognition database. http://web.mit.edu/torralba/www/indoor.html (2009)
106. Rajagopalan, R., Varshney, P.: Data-aggregation techniques in sensor networks: a survey. IEEE Commun. Surv. Tutor. **8**(4), 48 –63 (2006). doi:10.1109/COMST.2006.283821
107. Ramdas, T., Ang, L.M., Egan, G.: Fpga implementation of an integer mips processor in handel-c and its application to human face detection. In: TENCON 2004. 2004 IEEE Region 10 Conference Volume A, vol. 1, pp. 36–39 (2004). doi:10.1109/TENCON.2004.1414350
108. Rapantzikos, K., Avrithis, Y., Kollias, S.: Spatiotemporal saliency for event detection and representation in the 3d wavelet domain: Potential in human action recognition. In: Proceedings of the 6th ACM International Conference on Image and Video Retrieval (CIVR '07), pp. 294–301. ACM, New York (2007). doi:10.1145/1282280.1282326. URL http://doi.acm.org/10.1145/1282280.1282326
109. Reid, M.M., Millar, R.J., Black, N.D.: Second-generation image coding: An overview. ACM Comput. Surv. **29**(1), 3–29 (1997). doi:10.1145/248621.248622. URL http://doi.acm.org/10.1145/248621.248622

110. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: Proceedings of the 9th European Conference on Computer Vision - Volume Part I (ECCV'06), pp. 430–443. Springer, Berlin, Heidelberg (2006). doi:10.1007/11744023_34. URL http://dx.doi.org/10.1007/11744023_34
111. Said, A., Pearlman, W.A.: A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Trans. Circuits Syst. Video Technol. **6**(3), 243–250 (1996). doi:10.1109/76.499834. URL http://dx.doi.org/10.1109/76.499834
112. Sensirion AG: Sensirion Digital Humidity and Temperature Sensors (RH&T). http://www.sensirion.com/en/products/humidity-temperature/ (2011)
113. Shapiro, J.: Embedded image coding using zerotrees of wavelet coefficients. Trans. Signal Process. **41**(12), 3445–3462 (1993). doi:10.1109/78.258085. URL http://dx.doi.org/10.1109/78.258085
114. Skodras, A., Christopoulos, C., Ebrahimi, T.: The JPEG 2000 still image compression standard. IEEE Signal Process. Mag. **18**, 36–58 (2001)
115. Slepian, D., Wolf, J.: Noiseless coding of correlated information sources. IEEE Trans. Inf. Theory **19**(4), 471–480 (2006). doi:10.1109/TIT.1973.1055037. URL http://dx.doi.org/10.1109/TIT.1973.1055037
116. Stitt, G., Vahid, F., Nematbakhsh, S.: Energy savings and speedups from partitioning critical software loops to hardware in embedded systems. ACM Trans. Embed. Comput. Syst. **3**(1), 218–232 (2004). doi:10.1145/972627.972637. URL http://doi.acm.org/10.1145/972627.972637
117. Stojanovic, M.: Underwater acoustic communications: Design considerations on the physical layer. In: Fifth Annual Conference on Wireless on Demand Network Systems and Services, 2008 (WONS 2008). pp. 1–10 (2008). doi:10.1109/WONS.2008.4459349
118. Svensson, H.: Reconfigurable architectures for embedded systems. Ph.D. thesis, Lund University (2008)
119. Szeliski, R.: Image alignment and stitching: A tutorial. Found. Trends Comput. Graph. Vis. **2**(1), 1–104 (2006). doi:10.1561/0600000009. URL http://dx.doi.org/10.1561/0600000009
120. Taubman, D.: High performance scalable image compression with ebcot. Trans. Image Process. **9**(7), 1158–1170 (2000). doi:10.1109/83.847830. URL http://dx.doi.org/10.1109/83.847830
121. Telle, N., Luk, W., Cheung, R.: Customising hardware designs for elliptic curve cryptography. In: Pimentel, A., Vassiliadis, S. (eds.) Computer Systems: Architectures, Modeling, and Simulation. Lecture Notes in Computer Science, vol. 3133, pp. 274–283. Springer, Berlin (2004). doi:10.1007/978-3-540-27776-7_29. URL http://dx.doi.org/10.1007/978-3-540-27776-7_29
122. The International Society of Automation: ISA100, Wireless Systems for Automation. www.isa.org/isa100 (2008)
123. Todman, T., Constantinides, G., Wilton, S., Mencer, O., Luk, W., Cheung, P.: Reconfigurable computing: Architectures and design methods. IEE Proc. Comput. Digit. Tech. **152**(2), 193–207 (2005). doi:10.1049/ip-cdt:20045086
124. Treisman, A., Gelade, G.: A feature-integration theory of attention. Cogn. Psychol. **12**(1), 97–136 (1980)
125. Tsapatsoulis, N., Rapantzikos, K.: Wavelet based estimation of saliency maps in visual attention algorithms. In: Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part II (ICANN'06), pp. 538–547. Springer, Berlin (2006). doi:10.1007/11840930_56. URL http://dx.doi.org/10.1007/11840930_56
126. Tsapatsoulis, N., Rapantzikos, K., Pattichis, C.S.: An embedded saliency map estimator scheme: Application to video encoding. Int. J. Neural Syst. **17**(4), 289–304 (2007). URL http://dblp.uni-trier.de/db/journals/ijns/ijns17.html#TsapatsoulisRP07
127. Tsekoura, I., Selimis, G., Hulzink, J., Catthoor, F., Huisken, J., de Groot, H., Goutis, C.: Exploration of cryptographic ASIP designs for wireless sensor nodes. In: 2010 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), pp. 827–830 (2010). doi:10.1109/ICECS.2010.5724640

128. Tyson, Romas, A., Siti Intan, P., Adiono, T.: A pipelined double-issue mips based processor architecture. In: International Symposium on Intelligent Signal Processing and Communication Systems, 2009 (ISPACS 2009), pp. 583–586 (2009). doi:10.1109/ISPACS.2009.5383771
129. Urban, F., Follet, B., Chamaret, C., Le Meur, O., Baccino, T.: Medium spatial frequencies, a strong predictor of salience. Cogn. Comput. **3**, 37–47 (2011). doi:10.1007/s12559-010-9086-8. URL http://hal.inria.fr/inria-00628096
130. Viola, P., Jones, M.J.: Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004). doi:10.1023/B:VISI.0000013087.49260.fb. URL http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb
131. Wagner, R., Nowak, R., Baraniuk, R.: Distributed image compression for sensor networks using correspondence analysis and super-resolution. In: Proceedings of the 2003 International Conference on Image Processing, 2003 (ICIP 2003), vol. 1, pp. I–597–600 (2003). doi:10.1109/ICIP.2003.1247032
132. Wallace, G.K.: The jpeg still picture compression standard. Commun. ACM **34**(4), 30–44 (1991). doi:10.1145/103085.103089. URL http://doi.acm.org/10.1145/103085.103089
133. Walther, D., Koch, C.: 2006 special issue: Modeling attention to salient proto-objects. Neural Netw. **19**, 1395–1407 (2006). doi:10.1016/j.neunet.2006.10.001. URL http://dl.acm.org/citation.cfm?id=1219169.1219421
134. Warneke, B., Scott, M., Leibowitz, B., Zhou, L., Bellew, C., Chediak, J., Kahn, J., Boser, B., Pister, K.: An autonomous 16 mm3 solar-powered node for distributed wireless sensor networks. In: Proceedings of IEEE Sensors, 2002, vol. 2, pp. 1510–1515 (2002). doi:10.1109/ICSENS.2002.1037346
135. Wheeler, F., Pearlman, W.: Spiht image compression without lists. In: Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000, vol. 6 (ICASSP '00), vol.4, pp. 2047–2050 (2000). doi:10.1109/ICASSP.2000.859236
136. Winbond: Winbond W968D6DKA product page. http://www.winbond.com/hq/enu/ProductAndSales/ProductLines/MobileRAM/PseudoSRAM/W968D6DKA.htm (2008)
137. Wu, M., Chen, C.W.: Collaborative image coding and transmission over wireless sensor networks. EURASIP J. Appl. Signal Process. **2007**(1), 223–223 (2007). doi:10.1155/2007/70481. URL http://dx.doi.org/10.1155/2007/70481
138. Wyner, A.D., Ziv, J.: The rate-distortion function for source coding with side information at the decoder. IEEE Trans. Inf. Theory **22**, 1–10 (1976)
139. Xilinx: MicroBlaze Soft Processor Core. http://www.xilinx.com/tools/microblaze.htm (2012)
140. Xilinx: Spartan-3 datasheet. http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf (2012)
141. Xilinx: Xilinx Artix-7. http://www.xilinx.com/products/silicon-devices/fpga/artix-7/index.htm (2012)
142. Xilinx: Xilinx XPower. http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm (2012). Accessed Nov 2012
143. Zeidman, B.: Introduction to Verilog. Swiss Creek Publications (2000)
144. Zeidman, B.: Designing With FPGAs and CPLDs, 1st edn. CRC Press, Boca Raton (2002)
145. Zhang, J., Orlik, P., Sahinoglu, Z., Molisch, A., Kinney, P.: Uwb systems for wireless sensor networks. Proc. IEEE **97**(2), 313–331 (2009). doi:10.1109/JPROC.2008.2008786
146. ZigBee Alliance: ZigBee Standards. http://www.zigbee.org/Standards/Downloads.aspx (2007)
147. Zulkifli, M., Yudhanto, Y., Soetharyo, N., Adiono, T.: Reduced stall mips architecture using pre-fetching accelerator. In: International Conference on Electrical Engineering and Informatics, 2009 (ICEEI '09), vol. 02, pp. 611–616 (2009). doi:10.1109/ICEEI.2009.5254742
148. Zvikhachevskaya, A., Markarian, G., Mihaylova, L.: Quality of service consideration for the wireless telemedicine and e-health services. In: IEEE Wireless Communications and Networking Conference, 2009 (WCNC 2009), pp. 1–6 (2009). doi:10.1109/WCNC.2009.4917925

Index

L. Ang et al., *Wireless Multimedia Sensor Networks on Reconfigurable Hardware*,
DOI 10.1007/978-3-642-38203-1, © Springer-Verlag Berlin Heidelberg 2013