

```
1 #include "PWM.h"
2
3 namespace Hardware
4 {
5     /// <summary>
6     /// Constructeur
7     /// </summary>
8     /// <param name="pin">Pin</param>
9     PWM::PWM(Pin pin)
10    {
11        this->pin = pin;
12
13        // Check if PWM cape is loaded, if not load it
14        if (!CapeLoaded(PWM_CAPE)) { Write(SLOTS, PWM_CAPE); }
15
16        // Init the pin
17        basepath = OCP_PATH;
18        switch (pin)
19        {
20            case Hardware::PWM::P8_13:
21                if (!CapeLoaded(P8_13_CAPE)) { Write(SLOTS, P8_13_CAPE_LOAD); }
22                basepath.append(FindPath(P8_13_FIND));
23                break;
24            case Hardware::PWM::P8_19:
25                if (!CapeLoaded(P8_19_CAPE)) { Write(SLOTS, P8_19_CAPE_LOAD); }
26                basepath.append(FindPath(P8_19_FIND));
27                break;
28            case Hardware::PWM::P9_14:
29                if (!CapeLoaded(P9_14_CAPE)) { Write(SLOTS, P9_14_CAPE_LOAD); }
30                basepath.append(FindPath(P9_14_FIND));
31                break;
32            case Hardware::PWM::P9_16:
33                if (!CapeLoaded(P9_16_CAPE)) { Write(SLOTS, P9_16_CAPE_LOAD); }
34                basepath.append(FindPath(P9_16_FIND));
35                break;
36        }
37
38        // Get the working paths
39        dutypath = basepath + "/duty";
40        periodpath = basepath + "/period";
```

```
41     runpath = basepath + "/run";
42     polaritypath = basepath + "/polarity";
43
44     // Give Linux time to setup directory structure;
45     usleep(250000);
46
47     // Read current values
48     period = StringToNumber<int>(Read(periodpath));
49     duty = StringToNumber<int>(Read(dutypath));
50     run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
51     polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));
52
53     // calculate the current intensity
54     calcIntensity();
55 }
56
57
58 PWM::~PWM()
59 {
60 }
61
62 /// <summary>
63 /// Calculate the current intensity
64 /// </summary>
65 void PWM::calcIntensity()
66 {
67     if (polarity == Normal)
68     {
69         if (duty == 0) { intensity = 0.0f; }
70         else { intensity = (float)period / (float)duty; }
71     }
72     else
73     {
74         if (period == 0) { intensity = 0.0f; }
75         else { intensity = (float)duty / (float)period; }
76     }
77 }
78
79 /// <summary>
80 /// Set the intensity level as percentage
81 /// </summary>
82 /// <param name="value">floating value multiplication factor</param>
```

```
83 void PWM::SetIntensity(float value)
84 {
85     if (polarity == Normal)
86     {
87         SetDuty(static_cast<int>((value * duty) + 0.5));
88     }
89     else
90     {
91         SetPeriod(static_cast<int>((value * period) + 0.5));
92     }
93 }
94
95 /// <summary>
96 /// Set the output as a corresponding uint8_t value
97 /// </summary>
98 /// <param name="value">pixel value 0-255</param>
99 void PWM::SetPixelValue(uint8_t value)
100 {
101     if (period != 255) { SetPeriod(255); }
102     SetDuty(255 - value);
103     pixelvalue = value;
104 }
105
106 /// <summary>
107 /// Set the period of the signal
108 /// </summary>
109 /// <param name="value">period : int</param>
110 void PWM::SetPeriod(int value)
111 {
112     string valstr = NumberToString<int>(value);
113     Write(periodpath, valstr);
114     period = value;
115
116     calcIntensity();
117 }
118
119 /// <summary>
120 /// Set the duty of the signal
121 /// </summary>
122 /// <param name="value">duty : int</param>
123 void PWM::SetDuty(int value)
124
```

```
125     string valstr = NumberToString<int>(value);
126     Write(dutypath, valstr);
127     duty = value;
128
129     calcIntensity();
130 }
131
132 /// <summary>
133 /// Run the signal
134 /// </summary>
135 /// <param name="value">On or Off</param>
136 void PWM::SetRun(Run value)
137 {
138     int valInt = static_cast<int>(value);
139     string valstr = NumberToString<int>(valInt);
140     Write(runpath, valstr);
141     run = value;
142 }
143
144 /// <summary>
145 /// Set the polarity
146 /// </summary>
147 /// <param name="value">Normal or Inverted signal</param>
148 void PWM::SetPolarity(Polarity value)
149 {
150     int valInt = static_cast<int>(value);
151     string valstr = NumberToString<int>(valInt);
152     Write(runpath, valstr);
153     polarity = value;
154 }
155
156 /// <summary>
157 /// Find the current PWM path in the OCP.3 directory
158 /// </summary>
159 /// <param name="value">part a the path name</param>
160 /// <returns>Returns the first found value</returns>
161 string PWM::FindPath(string value)
162 {
163     auto dir = opendir(OCP_PATH);
164     auto entity = readdir(dir);
165     while (entity != NULL)
```

```
166     {
167         if (entity->d_type == DT_DIR)
168         {
169             string str = static_cast<string>(entity->d_name);
170             if (str.find(value) != string::npos) { return str; }
171         }
172         entity = readdir(dir);
173     }
174     return "";
175 }
176
177 }
```