SoC Linux 道場【其ノ七】
ユーザ回路の追加と動作確認

Ver.13.1

2017 年 8 月 Rev.2

目次

1. はじめに

前回の SoC Linux 道場【其ノ六】までの記事で、カスタム・ドライバを作成して実際に Helio ボードの I/O アクセスができるようになりました。

今回と次回は、Helio ボードの FPGA 上にモーターに対して PWM 制御するユーザ独自のハードウェアを追加して動作の確認を行います。

今回はまず、Helio ボードの LED の様子で PWM 制御できているかどうかを確かめます。

LED の PWM 制御は、スクリプト(コマンド・ライン)と CGI プログラムを使って行います。

尚、この資料の説明で使用している主な開発環境は以下の通りです。

【表 1.1】この資料の説明で使用している主な開発環境

項番	項目	内容
1	ホスト OS	Microsoft® Windows® 7 Professional sp1 日本語版(64 bit)
2	ゲスト OS	Vine Linux 6.5 x86_64 この資料では、Linux® 開発環境として、Vine Linux ディストリビューションを使用します。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
3	仮想 OS 実行環境	OS を仮想的に実行するための環境です。 この説明では、「VMware Player for Windows」と呼ばれるフリーウェア・ソフトを使用しています。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
4	クロス・コンパイラ	ターゲット(Helio)向けの実行イメージを生成するためのコンパイラです。 この説明では、Linaro から提供されている 32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーンを使用しています。 詳細については、SoC Linux 道場【其ノ参】を参照ください。
5	thttpd	組み込み機器では比較的使用されている Web サーバ・ソフトです。 「2-3. CGI による PWM 制御」を実行するために必要となります。 詳細については、SoC Linux 道場【其ノ四】を参照ください。
6	アルテラ Quartus® II	アルテラ FPGA のハードウェアを開発するためのツールです。 本説明書では、Quartus II バージョン v13.1 を使用しています。 ■ Quartus II 開発ソフトウェア https://www.altera.co.jp/products/design-software/fpga-design/quartus-ii/overview.html
7	Helio ボード	動作確認でターゲット・ボードとして使用する、 アルテラ Cyclone® V SoC を搭載したマクニカ Helio ボードです。 Helio には複数のリビジョンが存在しますが、この資料では、Rev1.2 または Rev1.3 を使用して動作確認を行っています。 ■ Helio ボード Rev1.2 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev12 ■ Helio ボード Rev1.3 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev13

2. ユーザ回路を追加

2-1. カスタム・ハードウェアの作成

カスタム・ドライバを作成して I/O アクセスができるようになりましたので、次にユーザ回路を追加します。ここでは、時間的な制御を行う PWM 制御回路を追加します。

【注記】

SoC Linux 道場【其ノ六】の「2-2. Helio のリファレンス・デザインの入手と確認」(1) ~ (5) で説明した、Quartus II のプロジェクト helio_ghrd_v13.1.zip がダウンロード済みで、任意のフォルダ（この例では、C:¥Temp¥Helio の下）に解凍済みであることを前提としています。

(1) Verilog-HDL により作成した PWM 制御回路（ファイル名： MOTOR_CNT.v）を**【リスト 2-1.1】**に示します。

別途記述済の MOTOR_CNT.v ファイルを用意していますので、これをダウンロードしてご利用ください。

このソース自体はそれほど難しいものではなく、主な機能としては以下の通りです。

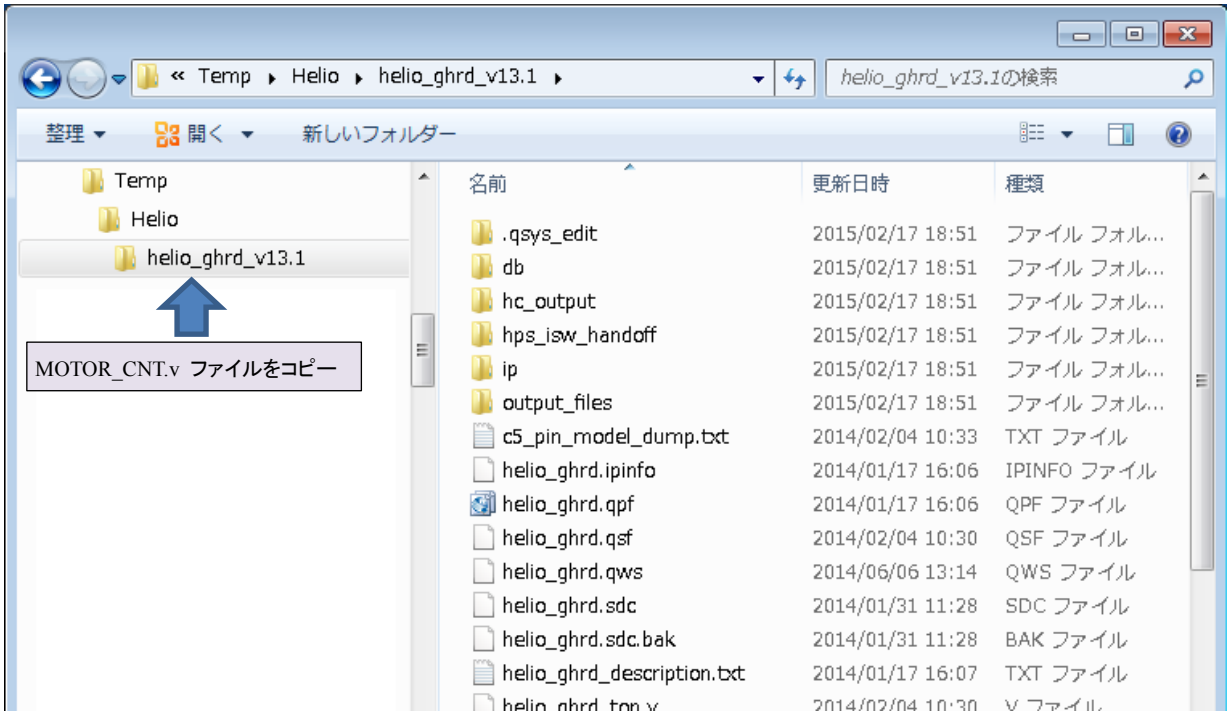
① 方向を変える入力（DIR_INPUT）により MOTOR_OUT の 2 ビットのいずれかに PWM の制御信号を出力する

② スピードは入力される値（SET_SW）によって 8 段階に切り替えられる

```
module MOTOR_CNT(input CLK, RESET, DIR_INPUT, input [2:0] SET_SW, output reg [1:0] MOTOR_OUT);
reg [20:0] tmp_cnt;
reg [21:0] set_signal;
reg DIR, EN;

parameter CNT_MAX = 2000000; // 20 ms

always @(posedge CLK or posedge RESET)
  if (RESET == 1'b1)
    tmp_cnt <= 12'h000;
  else if (tmp_cnt == (CNT_MAX - 1))
    tmp_cnt <= 12'h000;
  else
    tmp_cnt <= tmp_cnt + 12'h1;

always @(*)
begin
  case (SET_SW)
    3'b000 : set_signal = (CNT_MAX*1)/8;
    3'b001 : set_signal = (CNT_MAX*2)/8;
    3'b010 : set_signal = (CNT_MAX*3)/8;
    3'b011 : set_signal = (CNT_MAX*4)/8;
    3'b100 : set_signal = (CNT_MAX*5)/8;
    3'b101 : set_signal = (CNT_MAX*6)/8;
    3'b110 : set_signal = (CNT_MAX*7)/8;
    3'b111 : set_signal = (CNT_MAX*8)/8;
    default : set_signal = 0;
  endcase
end

always @(posedge CLK or posedge RESET)
  if (RESET == 1'b1)
    EN <= 1'b0;
  else
    begin
      if (tmp_cnt <= set_signal[20:0])
        EN <= 1'b1;
      else
        EN <= 1'b0;
    end

always @(posedge CLK or posedge RESET)
  if (RESET == 1'b1)
    MOTOR_OUT <= 2'b00;
  else
    begin
      if (DIR_INPUT == 1'b0)
        MOTOR_OUT <= {1'b0, EN};
      else
        MOTOR_OUT <= {EN, 1'b0};
    end
endmodule
```

【リスト 2-1.1】PWM 制御回路(ファイル名: MOTOR_CNT.v)

(2) この Verilog-HDL ファイル(MOTOR_CNT.v)を Quartus II の helio_ghrd_v13 プロジェクト・フォルダにコピーします。

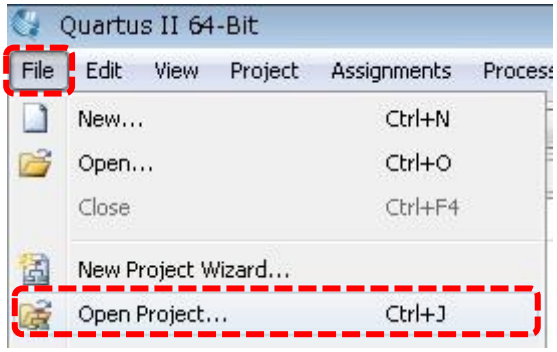


【図 2-1.1】MOTOR_CNT.v を helio_ghrd_v13 プロジェクト・フォルダにコピー

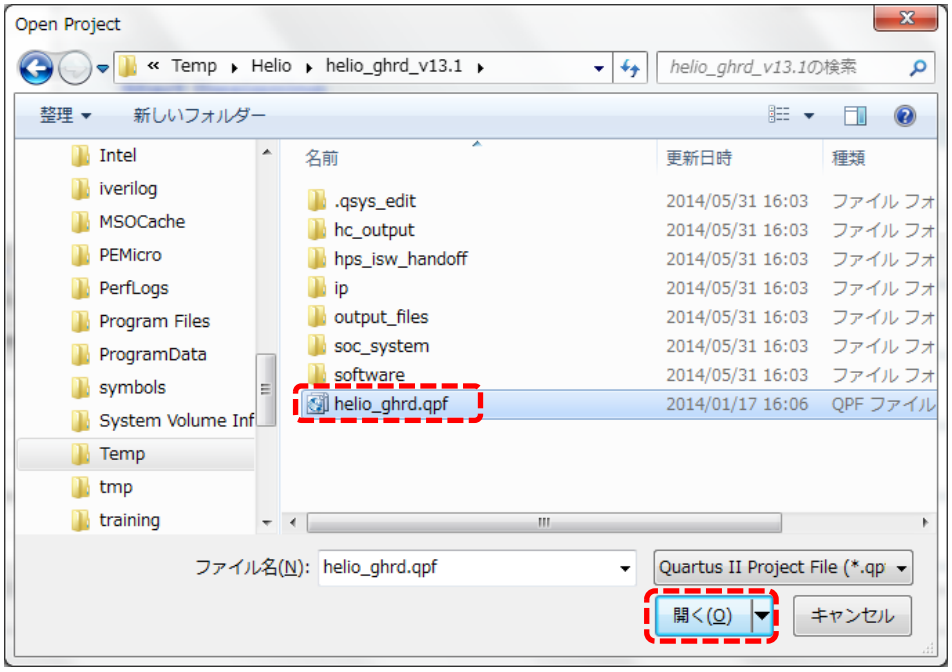
(3) PC のデスクトップにある Quartus II のアイコンをダブル・クリックして、Quartus II を起動します。

【図 2-1.2】Quartus II の起動

(4) Quartus II の File メニュー ⇒ Open Project… を選択して、解凍しておいたリファレンス・デザイン内の helio_ghrd.qpf ファイルを選択し、[開く]をクリックしてプロジェクトを Open します。

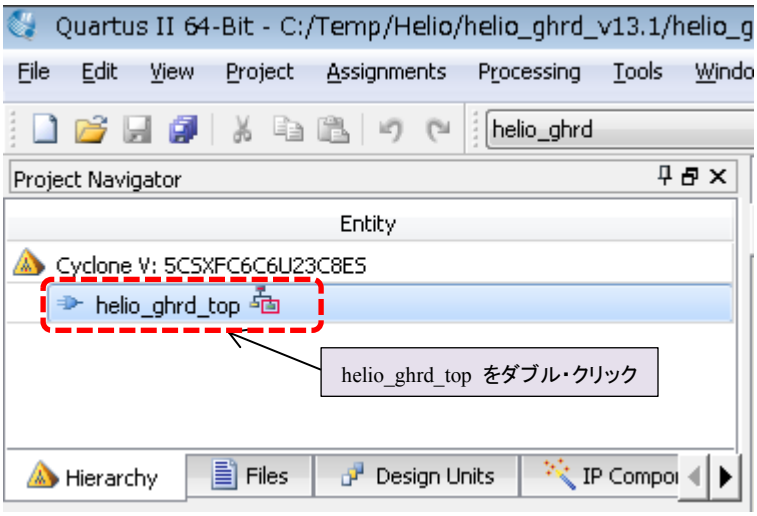


【図 2-1.3】File メニュー ⇒ Open Project… を選択



【図 2-1.4】リファレンス・デザインのプロジェクトのオープン

(5) helio_ghrd_top をダブル・クリックします。



【図 2-1.5】helio_ghrd_top をダブル・クリック

(6) トップ・ファイル(helio_ghrd_top.v)が開くので【リスト 2-1.2】のように MOTOR_CNT（赤色破線内）をインスタンスします。

```
module helio_ghrd_top (
//HPS-UART-----------------------------------------------------
    input           uart_rx,            //3.3V LV   //UART Receive
    output          uart_tx,            //3.3V      //UART Transmit

//HPS-I2C------------------------------------------------------
    inout           i2c_scl_hps,        //3.3V      //HPS I2C Clock output
    inout           i2c_sda_hps,        //3.3V      //HPS I2C Data Input/Output

//HPS-SPI-Bus--------------------------------------------------
    output          spi_csn,            //3.3V      //Slave Sel 0
    input           spi_miso,           //3.3V      //Master Input
    output          spi_mosi,           //3.3V      //Master Output
    output          spi_sck,            //3.3V      //Clock Output

//HPS-QSPI-Flash-----------------------------------------------
    output          qspi_clk,           //3.3V      //Clock
    inout   [ 3:0]  qspi_io,            //3.3V      //Data
    output          qspi_ss0,           //3.3V      //Select

//HPS-SD-Card-Flash--------------------------------------------
    output          sd_clk,             //3.3V      //
    inout           sd_cmd,             //3.3V      //
    inout   [ 3:0]  sd_dat,             //3.3V      //

//HPS-USB-OTG--------------------------------------------------
    input           usb_clk,            //3.3V      //Clock
    inout   [ 7:0]  usb_data,           //3.3V      //
    input           usb_nxt,            //3.3V      //
    input           usb_dir,            //3.3V      //
    output          usb_stp,            //3.3V      //

//HPS-Ethernet----10/100/1000----------------------------------
    output          enet_hps_gtx_clk,   //3.3V      //Gb Ethernet Clock
    output          enet_hps_mdc,       //3.3V      //MDIO Clock (TR=0)
    inout           enet_hps_mdio,      //3.3V      //MDIO Data (TR=0)
    input           enet_hps_rx_clk,    //3.3V      //Receive Data
    input           enet_hps_rx_dv,     //3.3V      //Receive Data Valid / Cont
    input   [ 3:0]  enet_hps_rxd,       //3.3V      //Receive Data
    output          enet_hps_tx_en,     //3.3V      //Transmit Data Enable / Cont
    output  [ 3:0]  enet_hps_txd,       //3.3V      //Transmit Data

//HPS-Trace----------------------------------------------------
    output          trace_clk_mic,      //3.3V      //
    output  [ 7:0]  trace_data,         //3.3V      //

//HPS-DDR3-400Mx32---------------------------------------------
    output  [14:0]  ddr3_hps_a,         //SSTL15    //Address
    output  [ 2:0]  ddr3_hps_ba,        //SSTL15    //Bank Address
    output          ddr3_hps_casn,      //SSTL15    //Column Address Strobe
    output          ddr3_hps_cke,       //SSTL15    //Clock Enable
```

（次ページへ続く）

（前ページからの続き）

```
    output          ddr3_hps_clk_n,     //SSTL15    //Diff Clock - Neg
    output          ddr3_hps_clk_p,     //SSTL15    //Diff Clock - Pos
    output          ddr3_hps_csn,       //SSTL15    //Chip Select
    output  [ 3:0]  ddr3_hps_dm,        //SSTL15    //Data Write Mask
    inout   [31:0]  ddr3_hps_dq,        //SSTL15    //Data Bus
    inout   [ 3:0]  ddr3_hps_dqs_n,     //SSTL15    //Diff Data Strobe - Neg
    inout   [ 3:0]  ddr3_hps_dqs_p,     //SSTL15    //Diff Data Strobe - Pos
    output          ddr3_hps_odt,       //SSTL15    //On-Die Termination Enable
    output          ddr3_hps_rasn,      //SSTL15    //Row Address Strobe
    output          ddr3_hps_resetn,    //SSTL15    //Reset
    output          ddr3_hps_wen,       //SSTL15    //Write Enable
    input           ddr3_hps_rzq,       //OCT_rzqin

//GPIOs-----------------------------------------------
   inout          hps_gpio09,
   inout          hps_gpio35,
   inout          hps_gpio41,
   inout          hps_gpio42,
   inout          hps_gpio43,
   inout          hps_gpio44,

//-------------------------------------------------------------------------//
////////////////////////////// FPGA Pins ///////////////////////////////////
//-------------------------------------------------------------------------//

//FPGA-GPLL-CLK-------------------------//X pins
    input           clk_100m_fpga,      //2.5V      //100 MHz (2nd copy to max)
    input           clk_50m_fpga,       //2.5V      //50MHz (2nd copy to max)
    input           clk_enet_fpga_p,    //LVDS      //125 MHz fixed

//FPGA-User-IO-----------------------------------------------
    input   [ 3:0]  user_dipsw_fpga,    //2.5V      //Dip SW pio
//    output  [ 3:0]  user_led_fpga,      //2.5V      //LED pio
    output  [ 3:0]  MOTOR_OUT,      //2.5V      //LED pio
    input   [ 2:0]  user_pb_fpga        //2.5V      //Push SW pio
);
//-------------------------------------------------------------------------//
//----------------------Logic description----------------------------------//
//-------------------------------------------------------------------------//


// internal wires and registers declaration
  wire [2:0] fpga_debounced_buttons;
  wire       hps_fpga_reset_n;

  wire [3:0] user_led_fpga;
  wire [1:0] motor_out_tmp;

    soc_system u0 (
        .clk_clk                            (clk_50m_fpga),
        .reset_reset_n                      (hps_fpga_reset_n),
        .hps_0_h2f_reset_reset_n            (hps_fpga_reset_n),
        .memory_mem_a                       (ddr3_hps_a),
```

（次ページへ続く）

（前ページからの続き）

```
        .memory_mem_ba                  (ddr3_hps_ba),
        .memory_mem_ck                  (ddr3_hps_clk_p),
        .memory_mem_ck_n                (ddr3_hps_clk_n),
        .memory_mem_cke                 (ddr3_hps_cke),
        .memory_mem_cs_n                (ddr3_hps_csn),
        .memory_mem_ras_n               (ddr3_hps_rasn),
        .memory_mem_cas_n               (ddr3_hps_casn),
        .memory_mem_we_n                (ddr3_hps_wen),
        .memory_mem_reset_n             (ddr3_hps_resetn),
        .memory_mem_dq                  (ddr3_hps_dq),
        .memory_mem_dqs                 (ddr3_hps_dqs_p),
        .memory_mem_dqs_n               (ddr3_hps_dqs_n),
        .memory_mem_odt                 (ddr3_hps_odt),
        .memory_mem_dm                  (ddr3_hps_dm),
        .memory_oct_rzqin               (ddr3_hps_rzq),
        .hps_io_hps_io_emac1_inst_TX_CLK    (enet_hps_gtx_clk),
        .hps_io_hps_io_emac1_inst_TXD0      (enet_hps_txd[0]),
        .hps_io_hps_io_emac1_inst_TXD1      (enet_hps_txd[1]),
        .hps_io_hps_io_emac1_inst_TXD2      (enet_hps_txd[2]),
        .hps_io_hps_io_emac1_inst_TXD3      (enet_hps_txd[3]),
        .hps_io_hps_io_emac1_inst_RXD0      (enet_hps_rxd[0]),
        .hps_io_hps_io_emac1_inst_MDIO      (enet_hps_mdio),
        .hps_io_hps_io_emac1_inst_MDC       (enet_hps_mdc),
        .hps_io_hps_io_emac1_inst_RX_CTL    (enet_hps_rx_dv),
        .hps_io_hps_io_emac1_inst_TX_CTL    (enet_hps_tx_en),
        .hps_io_hps_io_emac1_inst_RX_CLK    (enet_hps_rx_clk),
        .hps_io_hps_io_emac1_inst_RXD1      (enet_hps_rxd[1]),
        .hps_io_hps_io_emac1_inst_RXD2      (enet_hps_rxd[2]),
        .hps_io_hps_io_emac1_inst_RXD3      (enet_hps_rxd[3]),
        .hps_io_hps_io_qspi_inst_IO0        (qspi_io[0]),
        .hps_io_hps_io_qspi_inst_IO1        (qspi_io[1]),
        .hps_io_hps_io_qspi_inst_IO2        (qspi_io[2]),
        .hps_io_hps_io_qspi_inst_IO3        (qspi_io[3]),
        .hps_io_hps_io_qspi_inst_SS0        (qspi_ss0),
        .hps_io_hps_io_qspi_inst_CLK        (qspi_clk),
        .hps_io_hps_io_sdio_inst_CMD        (sd_cmd),
        .hps_io_hps_io_sdio_inst_D0         (sd_dat[0]),
        .hps_io_hps_io_sdio_inst_D1         (sd_dat[1]),
        .hps_io_hps_io_sdio_inst_CLK        (sd_clk),
        .hps_io_hps_io_sdio_inst_D2         (sd_dat[2]),
        .hps_io_hps_io_sdio_inst_D3         (sd_dat[3]),
        .hps_io_hps_io_usb1_inst_D0         (usb_data[0]),
        .hps_io_hps_io_usb1_inst_D1         (usb_data[1]),
        .hps_io_hps_io_usb1_inst_D2         (usb_data[2]),
        .hps_io_hps_io_usb1_inst_D3         (usb_data[3]),
        .hps_io_hps_io_usb1_inst_D4         (usb_data[4]),
        .hps_io_hps_io_usb1_inst_D5         (usb_data[5]),
        .hps_io_hps_io_usb1_inst_D6         (usb_data[6]),
        .hps_io_hps_io_usb1_inst_D7         (usb_data[7]),
        .hps_io_hps_io_usb1_inst_CLK        (usb_clk),
        .hps_io_hps_io_usb1_inst_STP        (usb_stp),
        .hps_io_hps_io_usb1_inst_DIR        (usb_dir),
        .hps_io_hps_io_usb1_inst_NXT        (usb_nxt),
```

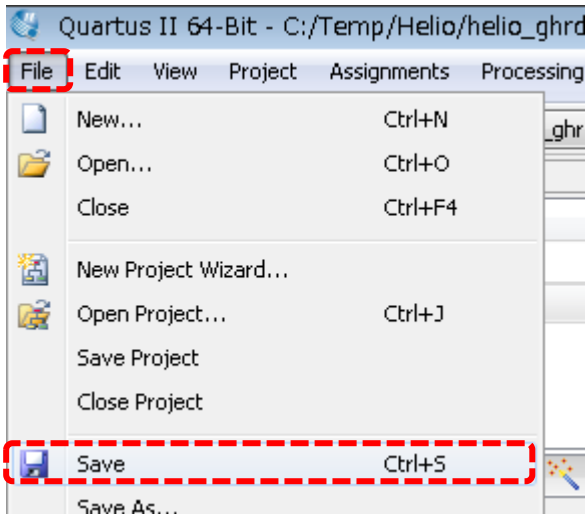
（次ページへ続く）

（前ページからの続き）

```
        .hps_io_hps_io_spim0_inst_CLK          (spi_sck),
        .hps_io_hps_io_spim0_inst_MOSI         (spi_mosi),
        .hps_io_hps_io_spim0_inst_MISO         (spi_miso),
        .hps_io_hps_io_spim0_inst_SS0          (spi_csn),
        .hps_io_hps_io_uart0_inst_RX           (uart_rx),
        .hps_io_hps_io_uart0_inst_TX           (uart_tx),
        .hps_io_hps_io_i2c0_inst_SDA           (i2c_sda_hps),
        .hps_io_hps_io_i2c0_inst_SCL           (i2c_scl_hps),
        .hps_io_hps_io_trace_inst_CLK          (trace_clk_mic),
        .hps_io_hps_io_trace_inst_D0           (trace_data[0]),
        .hps_io_hps_io_trace_inst_D1           (trace_data[1]),
        .hps_io_hps_io_trace_inst_D2           (trace_data[2]),
        .hps_io_hps_io_trace_inst_D3           (trace_data[3]),
        .hps_io_hps_io_trace_inst_D4           (trace_data[4]),
        .hps_io_hps_io_trace_inst_D5           (trace_data[5]),
        .hps_io_hps_io_trace_inst_D6           (trace_data[6]),
        .hps_io_hps_io_trace_inst_D7           (trace_data[7]),
        .hps_io_hps_io_gpio_inst_GPIO09        (hps_gpio09),
        .hps_io_hps_io_gpio_inst_GPIO35        (hps_gpio35),
        .hps_io_hps_io_gpio_inst_GPIO41        (hps_gpio41),
        .hps_io_hps_io_gpio_inst_GPIO42        (hps_gpio42),
        .hps_io_hps_io_gpio_inst_GPIO43        (hps_gpio43),
        .hps_io_hps_io_gpio_inst_GPIO44        (hps_gpio44),
        .led_pio_external_connection_export    (user_led_fpga),
        .dipsw_pio_external_connection_export  (user_dipsw_fpga),
        .button_pio_external_connection_export (fpga_debounced_buttons)
    );

    // Debounce logic to clean out glitches within 1ms
debounce debounce_inst (
  .clk                                  (clk_50m_fpga),
  .reset_n                              (hps_fpga_reset_n),
  .data_in                              (user_pb_fpga),
  .data_out                             (fpga_debounced_buttons)
);
  //defparam debounce_inst.WIDTH = 2;
  defparam debounce_inst.WIDTH = 3;
  defparam debounce_inst.POLARITY = "LOW";
  defparam debounce_inst.TIMEOUT = 50000;               // at 50Mhz this is a debounce time of 1ms
  defparam debounce_inst.TIMEOUT_WIDTH = 16;            // ceil(log2(TIMEOUT))

MOTOR_CNT u1(
  .CLK(clk_50m_fpga),
  .RESET(~user_pb_fpga[0]),
  .DIR_INPUT(~user_led_fpga[3]),
  .SET_SW(~user_led_fpga[2:0]),
  .MOTOR_OUT(motor_out_tmp)
  );

assign MOTOR_OUT = ~{user_led_fpga[3:2], motor_out_tmp};

endmodule
```

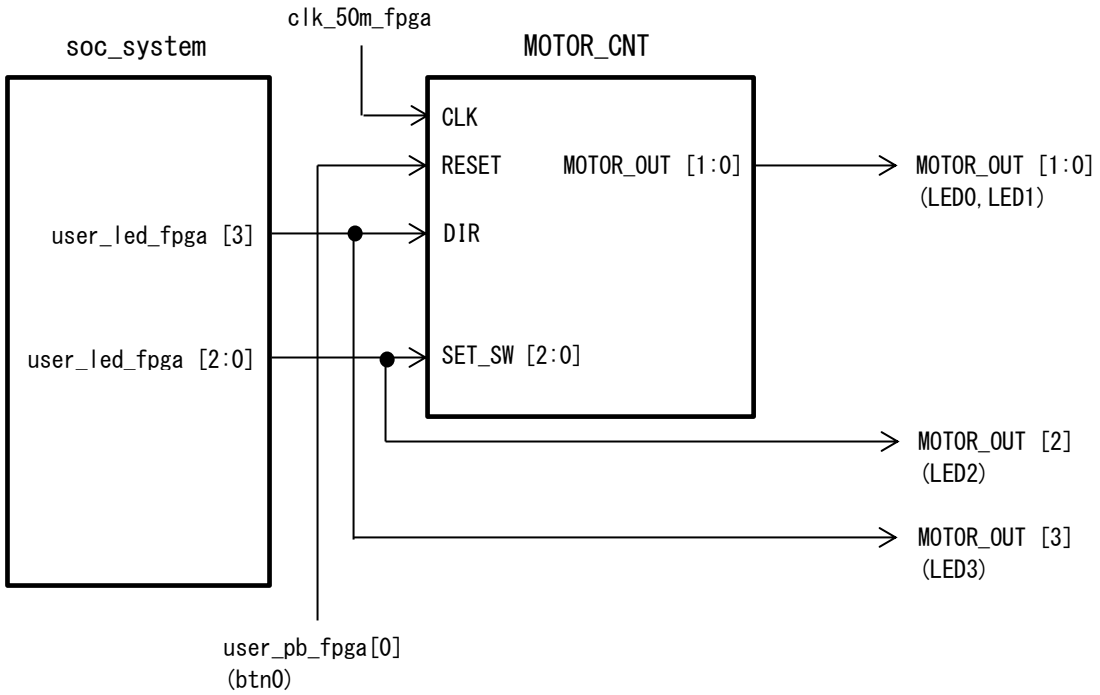
【リスト 2-1.2】トップ・ファイルへの MOTOR_CNT のインスタンス（helio_ghrd_top.v）

(7) インスタンスが終わったら File メニュー ⇒ Save でセーブします。



【図 2-1.6】File メニュー ⇒ Save でセーブ

(8) MOTOR_CNT の接続部分のブロック図としては下図のようになります。



【図 2-1.7】MOTOR_CNT のインスタンス部分のブロック図

(9) ユーザ回路の変更に伴い、出力名を user_led_fpga から MOTOR_OUT に変更したので、ピン・アサインを**【リスト 2-1.3】**のように変更します（helio_ghrd.qsf ファイルをエディタで開いて直接変更します）。

変更が終わったらファイルをセーブして、エディタを終了します。

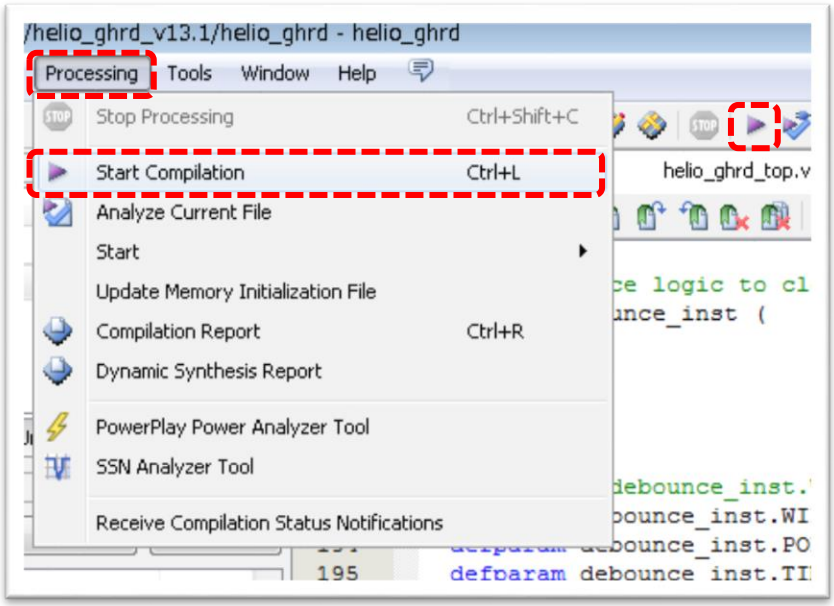
```
# set_location_assignment PIN_U9 -to user_led_fpga[0]
# set_location_assignment PIN_AD4 -to user_led_fpga[1]
# set_location_assignment PIN_V10 -to user_led_fpga[2]
# set_location_assignment PIN_AC4 -to user_led_fpga[3]

set_location_assignment PIN_U9 -to MOTOR_OUT[0]
set_location_assignment PIN_AD4 -to MOTOR_OUT[1]
set_location_assignment PIN_V10 -to MOTOR_OUT[2]
set_location_assignment PIN_AC4 -to MOTOR_OUT[3]
```

コメント・アウトする

左記の通り追加する

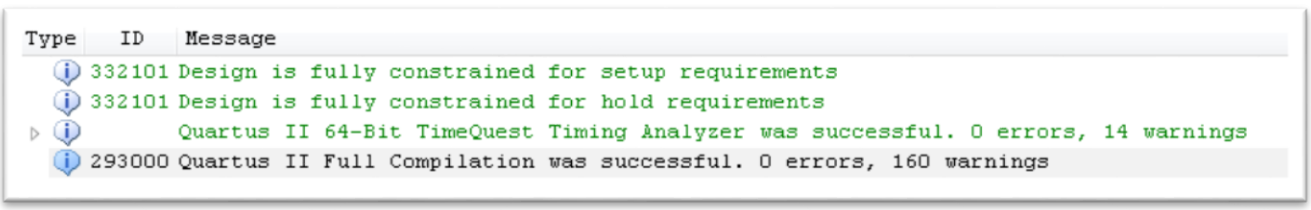
【リスト 2-1.3】ピン・アサインの変更（helio_ghrd.qsf の抜粋）

(10) Processing メニュー ⇒ Start Compilation または  ボタンをクリックして、コンパイルを実行します。



【図 2-1.8】コンパイルの実行

(11) コンパイルが成功すると、helio_ghrd_v13 プロジェクトの output_files フォルダに新しい helio_ghrd.sof ファイルが生成されているはずです。これを後程 Helio にダウンロードします。

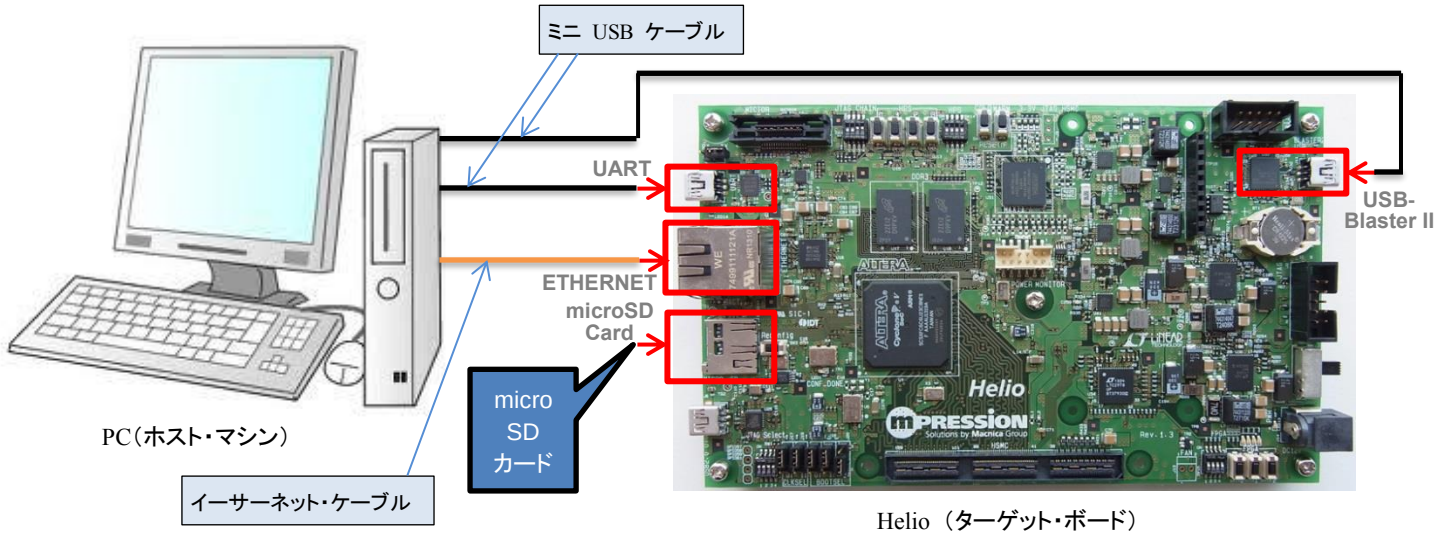


【図 2-1.9】コンパイルの成功

2-2. LED の PWM 制御(コマンド・ライン)

ここまで準備ができたら変更したデザインを Quartus II でコンパイルし、Programmer で生成された sof ファイルを Helio にダウンロードして、コマンド・ラインによる PWM 制御を確認します。

(1) Helio とホスト PC が下図のように接続されていることを確認します。



【図 2-2.1】PC と Helio の接続図(全体)

① Helio の UART コネクタとホスト PC の USB ポートをミニ USB ケーブルで接続します。

② Helio の USB-Blaster™ II コネクタとホスト PC の USB ポートをミニ USB ケーブルで接続します。

③ Helio の ETHERNET コネクタとホスト PC のイーサーネット・ポートをイーサーネット・ケーブルで接続します。

④ Helio の microSD カード・スロットに、SoC Linux 道場【其ノ六】で使用した Helio 用 Linux の microSD カードを取り付けます。

(2) Helio の電源を入れて Linux が起動したら、**ターゲット側(Helio)** で以下のように root でログインして、udhcpc コマンドを実行して、DHCP サーバから Helio に IP アドレスが付与されるようにします(下記の例では、192.168.1.238 が付与されています)。

【注記】

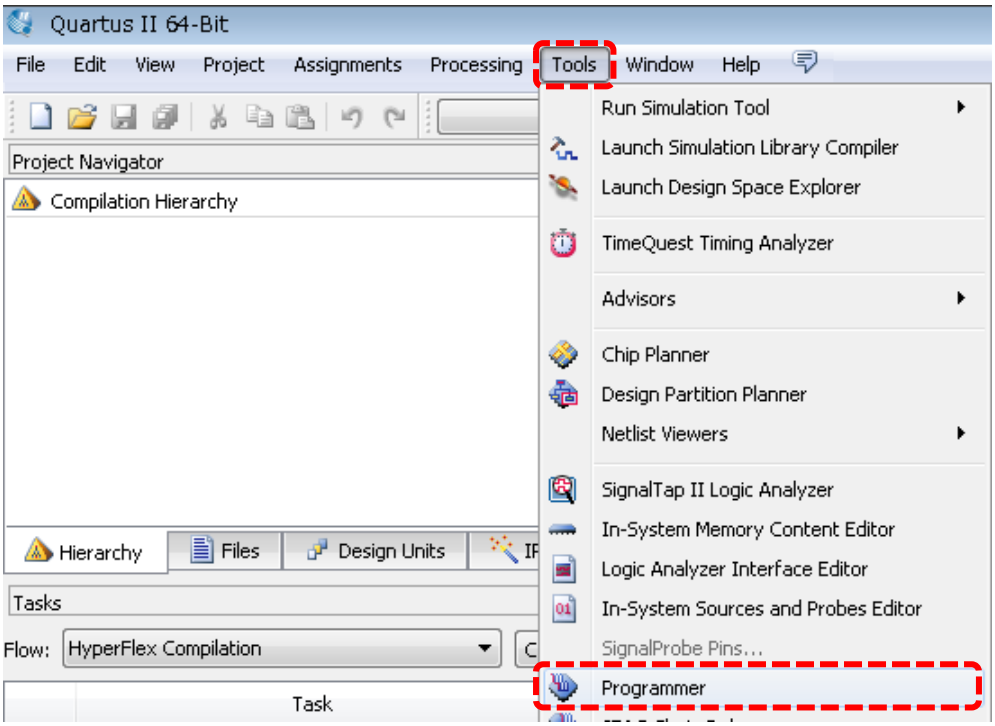
SoC Linux 道場【其ノ弐】で説明した DHCP サーバが設定されていて、使用できることを前提としています。

```
socfpga login: root
root@socfpga:~# udhcpc
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.238...
Lease of 192.168.1.238 obtained, lease time 21600
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

(3) PC のデスクトップにある Quartus II のアイコンをダブル・クリックして、Quartus II を起動します。
※ 既に Quartus II が起動済みである場合は、この手順はスキップしてください。

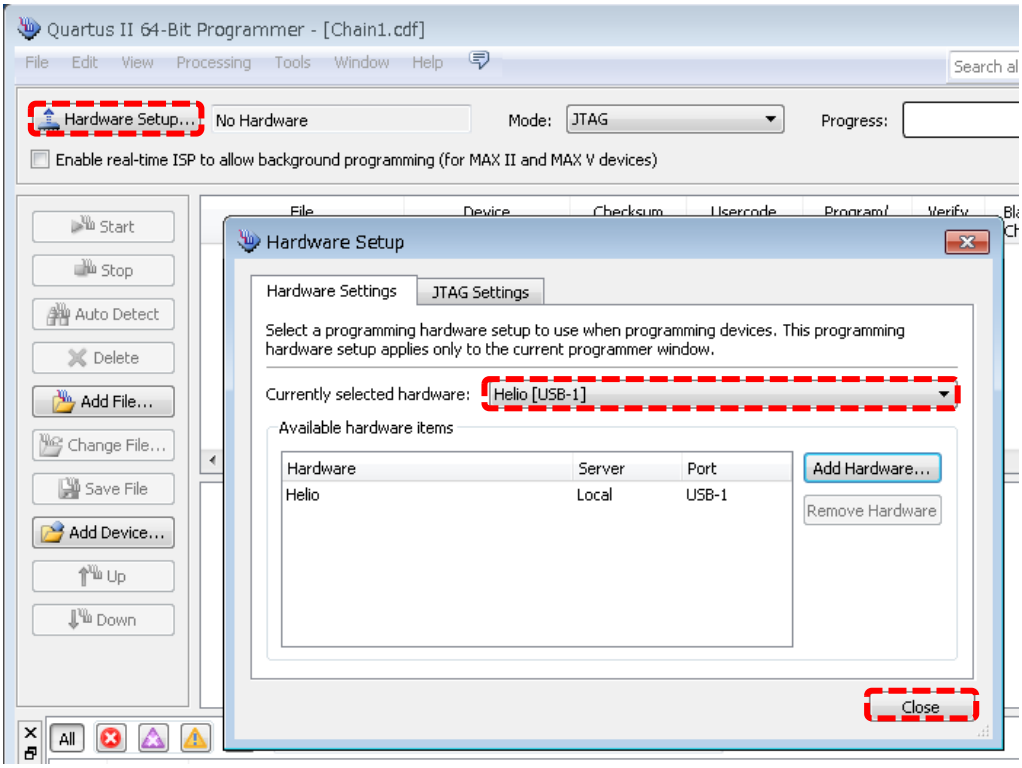
【図 2-2.2】Quartus II の起動

(4) Quartus II の Tools メニュー ⇒ Programmer を選択、または Programmer アイコン をクリックして Programmer を起動します。



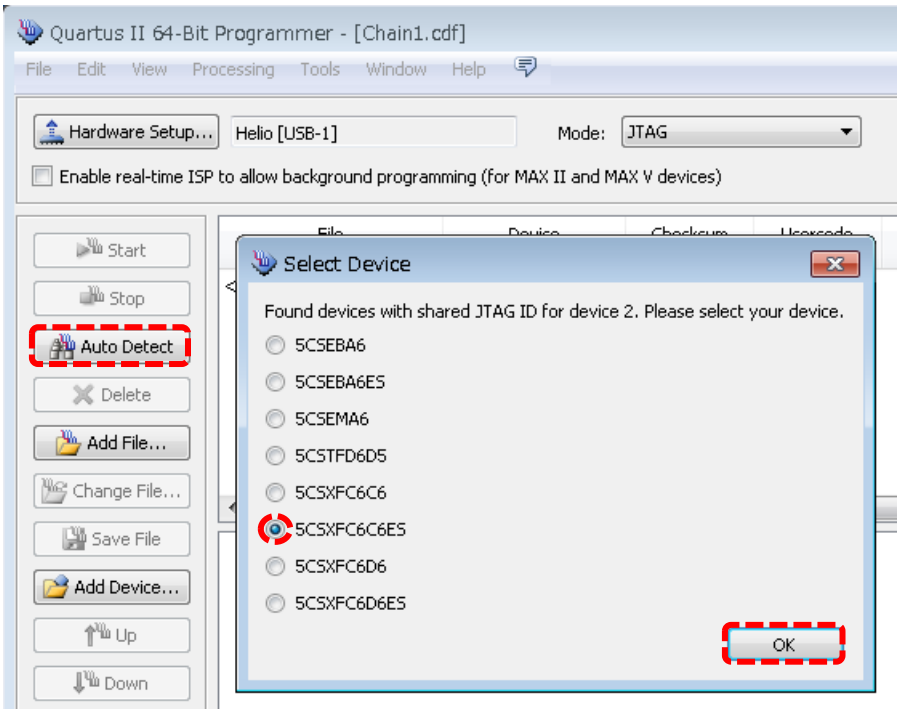
【図 2-2.3】Programmer の起動

(5) Programmer 内にある [Hardware Setup…] ボタンをクリックし、Currently selected hardware のプルダウンリストから Helio を選択します。選択したら [Close] をクリックします。



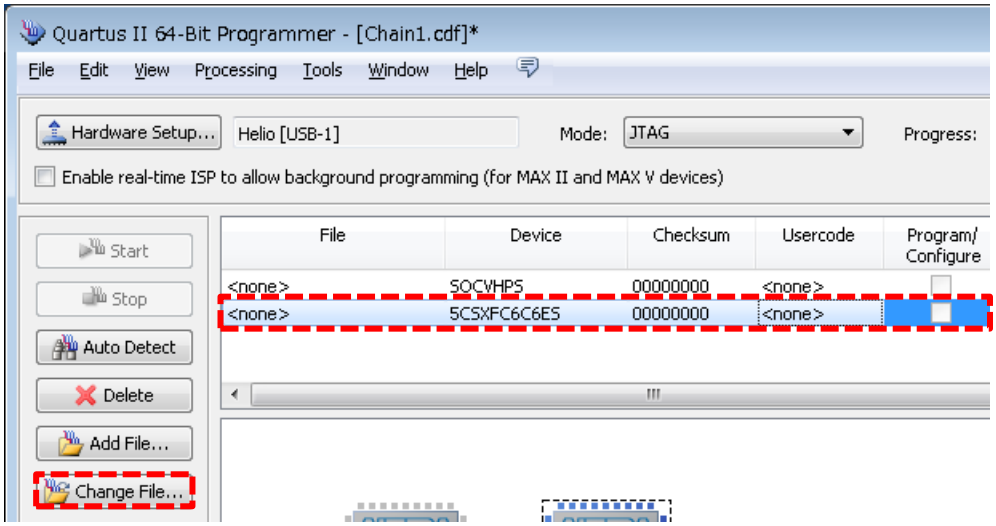
【図 2-2.4】Hardware Setup

(6) [Auto Detect] ボタンをクリックし、ボード上の JTAG チェインに接続されている FPGA を検出します。Select Device ウィンドウが現れたら “5CSXFC6C6ES” を選択し [OK] をクリックします。



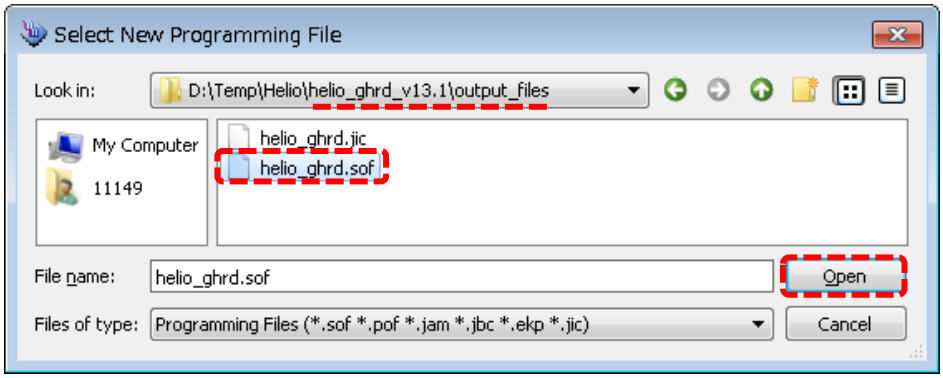
【図 2-2.5】デバイスの選択

(7) 5CSXFC6C6ES 上をクリックしてハイライトし [Change File] をクリックします。



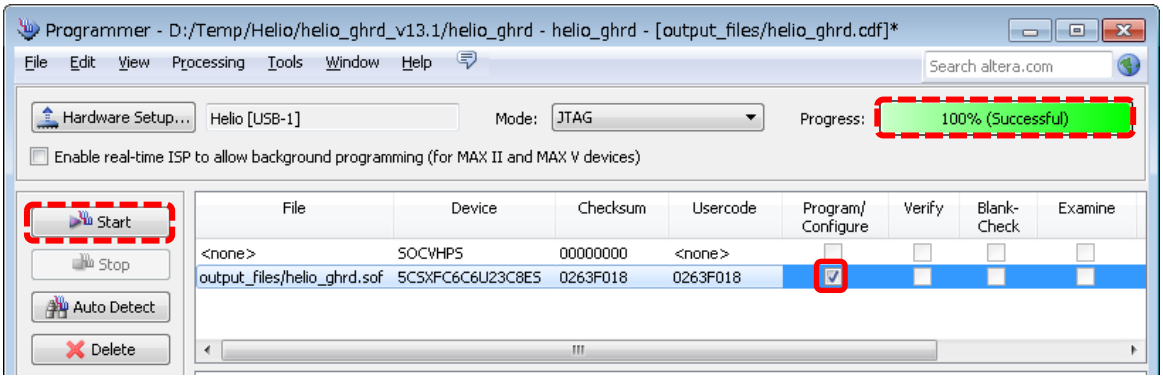
【図 2-2.6】[Change File] をクリック

(8) [Select New Programming File] ダイアログ・ボックスで、ダウンロード保存して解凍しておいた helio_ghrd_v13.1 フォルダの下の output_files フォルダをブラウズして、helio_ghrd.sof を選択し [Open] します。



【図 2-2.7】プログラミング・ファイルの選択

(9) Program/Configure にチェックを入れた後、[Start] ボタンをクリックしてコンフィギュレーションを行います。Progress バーに「100% (Successful)」と表示されればプログラミングは完了です。



【図 2-2.8】リファレンス・デザインによるコンフィギュレーションの実行

(10) 次に **ターゲット側(Helio)**から PWM 制御のための信号を SoC Linux 道場【其ノ六】の「2-5. デバイス・ファイルの作成とドライバ・アクセス」の (3) で作成したデバイス・ファイル /home/root/led0~3 に送ります。

【リスト 2-2.1】のようなスクリプトを作成しておくと便利です。

別途記述済の pwm_set.sh スクリプト・ファイルを用意しましたので、これをダウンロードして SoC Linux 道場【其ノ弐】で説明した Samba サーバ経由で、Windows から Vine Linux のホーム・ディレクトリ（/home/tori）にコピーしてご利用ください。

【注記】

Linux 上で使用する pwm_set.sh スクリプト・ファイルは、**文字コードを「UTF-8」**、**改行コードを「LF のみ」**で作成する必要があります。別途記述済の pwm_set.sh スクリプト・ファイルは、文字コード「UTF-8」、改行コード「LF のみ」で作成してありますが、ご自身で pwm_set.sh スクリプト・ファイルを作成する場合は注意してください。

```
#!/bin/sh

echo "ARG Value = $1"
LED0=`expr $1 ¥% 2`
LED1=`expr $1 ¥/ 2 ¥% 2`
LED2=`expr $1 ¥/ 4 ¥% 2`
LED3=`expr $1 ¥/ 8 ¥% 2`

echo $LED0 > ~/led0
echo $LED1 > ~/led1
echo $LED2 > ~/led2
echo $LED3 > ~/led3

echo "LED0 = $LED0"
echo "LED1 = $LED1"
echo "LED2 = $LED2"
echo "LED3 = $LED3"
```

【リスト 2-2.1】PWM 制御のスクリプト(pwm_set.sh)

(11) 上記の pwm_set.sh スクリプト・ファイルを NFS 経由で Helio 上に運ぶための準備として、**ターゲット側(Helio)**から**ホスト側(Vine Linux)**のマウント先 /opt/Helio に pwm_set.sh をコピーしておきます。

ホスト側(Vine Linux) から以下のコマンドを実行します。

```
[tori@Vine65 ~]$ cp pwm_set.sh /opt/Helio
```

(12) Helio の SD カードの /bin の下にもともと入っている busybox を使って、**ホスト側（Vine Linux）**の /opt/Helio を /mount_dir にマウントします。
これにより、上記 (11) で /opt/Helio にコピーした pwm_set.sh を Helio の /mount_dir から直接アクセスできるようになります。

また、後で説明する「2-3. CGI による PWM 制御」を実行するためには、SoC Linux 道場【其ノ四】で説明した thttpd という Web サーバ・ソフトを使用します。
SoC Linux 道場【其ノ四】の「2-2. thttpd のクロス・コンパイルとインストール」を一通り実施している場合は、thttpd のファイル（thttpd、thttpd.conf、index.html、printenv.cgi）が /opt/Helio に既にコピーされているはずなので、これらのファイルも Helio の /mount_dir から直接アクセスできるようになります。

ターゲット側（Helio） で以下のコマンドを実行します。

【注記】

※1. SoC Linux 道場【其ノ弐】で説明した、NFS サーバが使用できることを前提としています。

※2. thttpd のファイル（thttpd、thttpd.conf、index.html、printenv.cgi）が、**ホスト側（Vine Linux）**の /opt/Helio ディレクトリにコピーされていない場合は、SoC Linux 道場【其ノ四】の「2-2. thttpd のクロス・コンパイルとインストール」の 20 ページ (8) まで実行して thttpd のファイルを /opt/Helio ディレクトリにコピーしてから、これ以降のステップを実行してください。

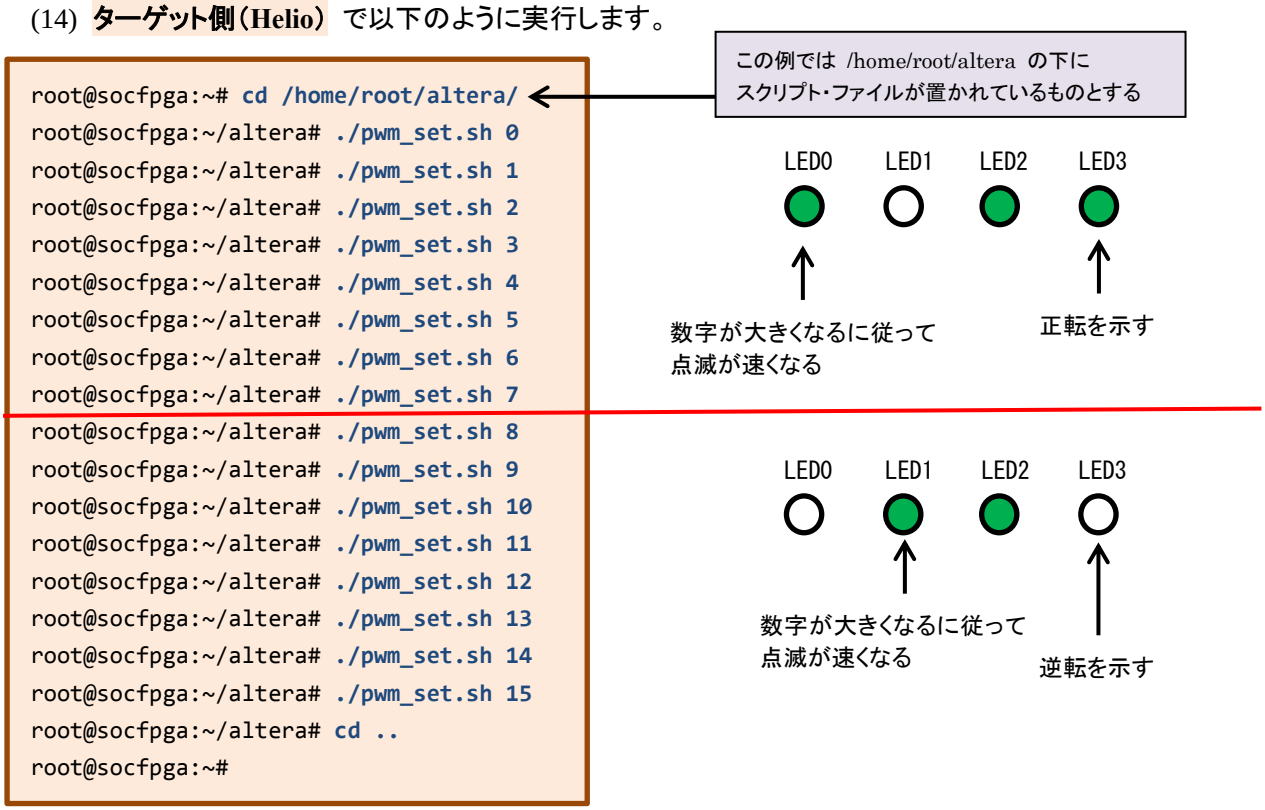
```
root@socfpga:~# busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
root@socfpga:~# cp /mount_dir/pwm_set.sh /home/root/altera/
```

(13) **ターゲット側（Helio）** で以下の insmod コマンドを実行して、SoC Linux 道場【其ノ六】ですでに作成してマウントされている **helio_gpio.ko** ドライバをロードします。

```
root@socfpga:~# insmod /mount_dir/helio_gpio.ko
Device registered successfully, Major No. = 252
```

(14) **ターゲット側（Helio）** で以下のように実行します。

```
root@socfpga:~# cd /home/root/altera/
root@socfpga:~/altera# ./pwm_set.sh 0
root@socfpga:~/altera# ./pwm_set.sh 1
root@socfpga:~/altera# ./pwm_set.sh 2
root@socfpga:~/altera# ./pwm_set.sh 3
root@socfpga:~/altera# ./pwm_set.sh 4
root@socfpga:~/altera# ./pwm_set.sh 5
root@socfpga:~/altera# ./pwm_set.sh 6
root@socfpga:~/altera# ./pwm_set.sh 7
root@socfpga:~/altera# ./pwm_set.sh 8
root@socfpga:~/altera# ./pwm_set.sh 9
root@socfpga:~/altera# ./pwm_set.sh 10
root@socfpga:~/altera# ./pwm_set.sh 11
root@socfpga:~/altera# ./pwm_set.sh 12
root@socfpga:~/altera# ./pwm_set.sh 13
root@socfpga:~/altera# ./pwm_set.sh 14
root@socfpga:~/altera# ./pwm_set.sh 15
root@socfpga:~/altera# cd ..
root@socfpga:~#
```



【図 2-2.9】PWM 制御のスクリプトを使用した LED の状態

① **「正転時」**（数値 **0~7** を与えた場合）は、**LED0 が点滅**し、**LED1 は消灯**し続けます。

② **「逆転時」**（数値 **8~15** を与えた場合）は、**LED1 が点滅**し、**LED0 は消灯**し続けます。

③ **LED3** は回転方向を示しており、**「正転時」は点灯**し、**「逆転時」は消灯**します。

④ **LED2** には user_led_fpga[2]（ビット 2）の信号がそのまま入っているので、与える数値が、

- **0~3** または **8~11** の場合は **点灯**
- **4~7** または **12~15** の場合は **消灯**

します。

もしうまく行かない場合には、HDL のソースやピン・アサインが正確に行われているか、また qsf ファイルなどの中身をよくチェックしてみましょう。

2-3. CGI による PWM 制御

スクリプトでコントロールできたので、今度は CGI プログラムを作成して、ブラウザからコントロールしてみます。

(1) CGI プログラムのソース・コード(ファイル名: pwm_cnt.c)を次の**【リスト 2-3.1】**に示します。

ソース・コードは Leafpad エディタ等を使用して一から書くこともできますが、大変なので別途記述済の pwm_cnt.c ソース・コードを用意しました。

この記述済 pwm_cnt.c ソース・コードをダウンロードして、SoC Linux 道場【其ノ弐】で説明した Samba サーバ経由で、Windows から Vine Linux のホーム・ディレクトリ(/home/tori)にコピーしてご利用ください。

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int i, j, last_result0, last_result1;
  int led_fd[4];
  char *env;

static char *LED[] = {
  "/home/root/led0",
  "/home/root/led1",
  "/home/root/led2",
  "/home/root/led3"
};

static char *MES0[] = {
  "Pos",
  "Neg"
};

static char *MES1[] = {
  "Speed0",
  "Speed1",
  "Speed2",
  "Speed3",
  "Speed4",
  "Speed5",
  "Speed6",
  "Speed7"
};
  printf("Content-type: text/html¥n¥n");
  printf("<html><body><p><big>Motor control CGI sample</big></p>¥n");
  printf("<form action=¥"pwm_cnt.cgi¥" method=¥"get¥">¥n");
  env = (char *)getenv("QUERY_STRING") ;
  if (env == NULL) env = "Pos Speed0";
//printf("<p>%s</p>¥n", env);
```

(次ページへ続く)

（前ページからの続き）

```
  for (i = 0; i <= 3; i++) {
    if ((led_fd[i] = open(LED[i], O_WRONLY)) == -1) {
        perror("open:");
        return 1;
    }
  }
  for (j = 0; j <= 1; j++) {
    printf("%s<input type=¥"radio¥" name=¥"Motor_DIR¥" value=¥"%s¥"", MES0[j], MES0[j]);
    if (strstr(env,MES0[j])) {
      printf(" checked>¥n");
      last_result0 = j;
    }
    else {
      printf(">¥n");
    }
  }
  printf("<BR>¥n");
  for (j = 0; j <= 7; j++) {
    printf("%s<input type=¥"radio¥" name=¥"Motor_SPEED¥" value=¥"%s¥"", MES1[j], MES1[j]);
    if (strstr(env,MES1[j])) {
      printf(" checked>¥n");
      last_result1 = j;
    }
    else {
      printf(">¥n");
    }
  }
  printf("<BR>¥n");
  if (last_result0 == 1)
    write (led_fd[3],"1",1);
  else
    write (led_fd[3],"0",1);
  for (i = 0; i <= 2; i++) {
    if (last_result1 & (0x01 << i))
      write (led_fd[i],"1",1);
    else
      write (led_fd[i],"0",1);
  }
  printf("<input type=¥"submit¥" value=¥"submit¥"><br>¥n");
  printf("</form></body></html>¥n");
  return 0;
}
```

【リスト 2-3.1】PWM 制御 CGI プログラム（ファイル名： pwm_cnt.c）

(2) **ホスト側（Vine Linux）** から以下のコマンドで、pwm_cnt.c のクロス・コンパイルを実行します。

```
[tori@Vine65 ~]$ arm-linux-gnueabihf-gcc pwm_cnt.c -o pwm_cnt.cgi
```

(3) pwm_cnt.cgi が生成されているはずなので、それを NFS 経由で Helio 上に運ぶための準備として、**ターゲット側（Helio）**から**ホスト側（Vine Linux）**のマウント先 /opt/Helio に pwm_cnt.cgi をコピーしておきます。

ホスト側（Vine Linux） から以下のコマンドを実行します。

```
[tori@Vine65 ~]$ cp pwm_cnt.cgi /opt/Helio/
```

(4) Helio の SD カードの /bin の下にもともと入っている busybox を使って、**ホスト側（Vine Linux）**の /opt/Helio を /mount_dir にマウントします。**ターゲット側（Helio）** で以下のコマンドを実行します。これにより、上記 (3) で /opt/Helio にコピーした pwm_cnt.cgi を Helio の /mount_dir から直接アクセスできるようになります。

※ 前出の「2-2. LED の PWM 制御（コマンド・ライン）」の (12) で、すでにマウント済みであれば、この手順はスキップしてください。

【注記】

SoC Linux 道場【其ノ弐】で説明した、NFS サーバが使用できることを前提としています。

```
root@socfpga:~# busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
```

(5) **ターゲット側（Helio）** で thttpd の設定を行います。まず NFS 経由で見えている 4 つのファイル（thttpd、thttpd.conf、index.html、printenv.cgi）を root ディレクトリにコピーしておきます。

※ SoC Linux 道場【其ノ六】の「2-6. LED の CGI 制御」(5) で、すでにコピー済みであれば、この手順はスキップしてください。

```
root@socfpga:~# cp /mount_dir/thttpd .
root@socfpga:~# cp /mount_dir/thttpd.conf .
root@socfpga:~# cp /mount_dir/index.html .
root@socfpga:~# cp /mount_dir/printenv.cgi .
```

最後の「. 」（ドット）を忘れないこと

(6) **ターゲット側（Helio）** で以下のように必要なユーザやファイルを作成します。

※ SoC Linux 道場【其ノ六】の「2-6. LED の CGI 制御」(6) で、すでに実施済みであれば、この手順はスキップしてください。

```
root@socfpga:~# adduser httpd
Changing password for httpd
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Bad password: too simple.

Warning: weak password (continuing).
Re-enter new password:
Password changed.
root@socfpga:~# mkdir /home/httpd/html
root@socfpga:~# mkdir /home/httpd/html/cgi-bin
root@socfpga:~# cp index.html /home/httpd/html/
root@socfpga:~# chmod 644 /home/httpd/html/index.html
root@socfpga:~# cp printenv.cgi /home/httpd/html/cgi-bin
root@socfpga:~# chmod 755 /home/httpd/html/cgi-bin/printenv.cgi
```

SoC Linux 道場【其ノ四】で実施済みであれば、エラーが出ますが無視して問題ありません

適当なパスワード（例えば、toriumi）を入力します（非表示）

SoC Linux 道場【其ノ四】で実施済みであれば、エラーが出ますが無視して問題ありません

(7) **ターゲット側（Helio）** で pwm_cnt.cgi ファイルを /home/httpd/html/cgi-bin ディレクトリへコピーしてから、chmod コマンドで実行権限を与えます。

```
root@socfpga:~# cp /mount_dir/pwm_cnt.cgi /home/httpd/html/cgi-bin
root@socfpga:~# chmod 755 /home/httpd/html/cgi-bin/pwm_cnt.cgi
```

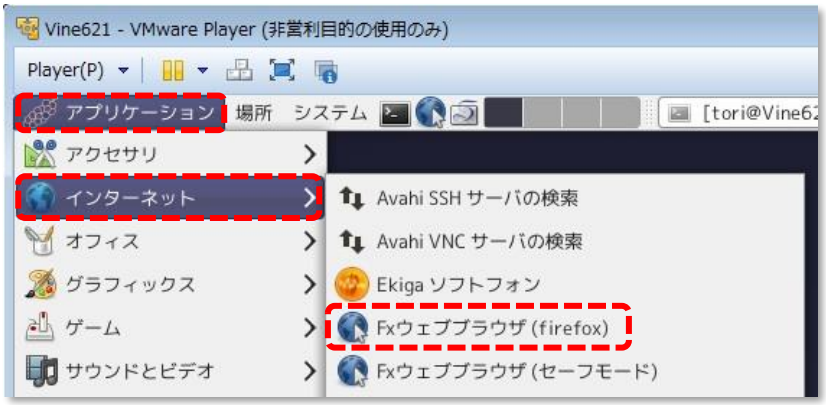
(8) **ターゲット側（Helio）** で thttpd を以下のコマンドで起動します。その後、ps コマンドでプロセスが起動していることを確認します。

```
root@socfpga:~# ./thttpd -C thttpd.conf
root@socfpga:~# ps
  PID USER       VSZ STAT COMMAND
    1 root      1312 S    init [5]
    2 root         0 SW   [kthreadd]
    3 root         0 SW   [ksoftirqd/0]

    （途中省略）

  704 httpd     2388 S    ./thttpd -C thttpd.conf
  705 root      1944 R    ps
root@socfpga:~#
```

thttpd を起動

プロセスが起動していることを確認

(9) **ホスト側（Vine Linux）** で Web ブラウザ（Firefox）を起動します。



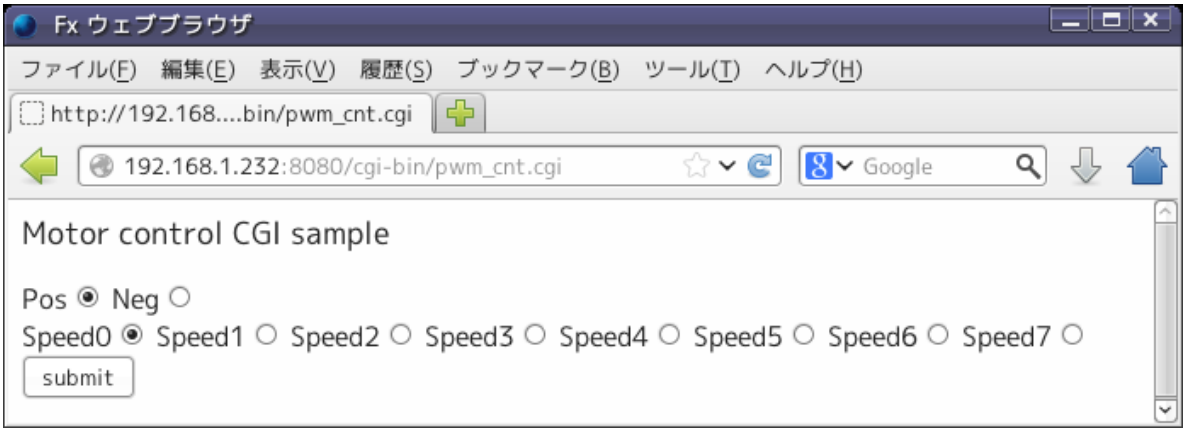
【図 2-3.1】Web ブラウザ（Fire Fox）を起動

(10) 例えば DHCP サーバから付与された Helio の IP アドレスが「192.168.1.238」の場合は、

```
http://192.168.1.238:8080/cgi-bin/pwm_cnt.cgi
```

を URL に指定して、下図のようなブラウザが現れて PWM 制御できるようになります。

'Pos' にチェックを入れると「正転」、'Neg' にチェックを入れると「逆転」になります。また各 'Speed' にチェックを入れると、数字が大きくなるに従って LED の点滅速度が速くなります。[submit] ボタンをクリックすると設定した PWM 制御を実行します。



【図 2-3.2】PWM 制御 CGI

改版履歴

Revision	年月	概要
1	2015 年 3 月	新規作成
1.1	2015 年 3 月	アルテラ社の Web サイトのリニューアルに伴う URL 変更
2	2017 年 8 月	① Sourcery CodeBench Lite Edition for ARM GNU/Linux の配布終了に伴い、クロス・コンパイル環境として Linaro Toolchain を使用する説明に変更 ② ゲスト OS を Vine Linux 6.2.1 i686 から Vine Linux 6.5 x86_64 に変更