课程设计项目

A 档

1 基于异构平台的交通标志识别系统

当今智能交通系统快速发展，交通标志识别系统作为其重要模块之一也被研究开发人员所重视并被广泛应用于自动驾驶等领域。

设计实现交通标志识别系统，并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 PYNQ-Z2 上板实现，检查分析系统设计的合理性并进行效果分析。

2 基于异构平台的交通信号灯智能识别系统

目前智慧交通和辅助驾驶是改善城市交通网络的关键所在，辅助驾驶将人工智能技术运用到城市交通运行中，协助交通部门，车辆驾驶员以及行人等缓解城市交通压力。交通信号灯作为道路交通中极其重要的标识之一，交通信号灯的识别对于智慧交通和辅助驾驶的重要性不言而喻。

设计实现红绿灯智能识别系统（识别灯向和灯色），并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 PYNQ-Z2 上板实现，检查分析系统设计的合理性并进行效果分析。

3 基于异构平台的面部微表情识别系统

微表情无处不在，对面部细微变化的识别存在于社会中的诸多领域。如在公安刑侦领域中，审讯人员可观察犯罪嫌疑人的微表情变化，更精确地判断证词的真实性。但由于微表情的动作幅度较小且持续时间短，其判断结果往往具有主观多样性。

设计实现面部微表情识别系统，并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 PYNQ-Z2 上板实现，检查分析系统设计的合理性并进行效果分析。

4 基于异构平台的人脸识别系统：

近年来，人脸识别在支付、安保、机器人等领域得到了广泛应用，已成为计算机视觉领域的研究热点。人脸识别需要检测、对齐和识别等步骤，在 PC 机上

开发算力相对足够，但是较难部署在嵌入式终端，需要综合考虑实时性、准确率、算力、功耗、成本、便携性、开发难易程度等因素。

设计实现人脸识别系统，在嵌入式终端实现结合深度学习的人脸识别——实时视频输入、人脸识别、显示结果输出。使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 PYNQ-Z2 上板实现，检查分析系统设计的合理性并进行效果分析。

B 档

5 基于异构平台的汽车自动防撞系统

自动驾驶技术作为目前人工智能技术一直研究的热点，是庞大而繁复的工程，为了一窥软硬件协同设计在自动驾驶领域的应用，利用本学期已经学习的软硬件协同课程设计的知识研究自动驾驶中的不可或缺的子系统----汽车自动防撞系统（包含行人检测）。

设计实现汽车自动防撞系统，并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 FPGA 上板实现，仿真系统设计的合理性。

6 基于异构平台的实现汽车自适应巡航控制系统

汽车自适应巡航控制系统是在定速巡航控制系统基础上发展起来的新一代汽车先进驾驶辅助系统。它将汽车定速巡航控制系统（Cruise Control System，CCS）和车辆前向撞击报警系统（Forward Collision Warning System，FCWS）有机结合起来，既有定速巡航控制系统的全部功能，还可以通过车载雷达等传感器监测汽车前方的道路交通环境，一旦发现当前行驶车道的前方有其他前行车辆，将根据本车和前车之间的相对距离及相对速度等信息，对车辆进行纵向速度控制，使本车与前车保持安全距离行驶，避免追尾事故发生。

设计实现汽车自适应巡航控制系统，并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 FPGA 上板实现，仿真系统设计的合理性。

C 档

7 基于异构平台的交通信号灯灯色识别系统

设计实现红绿灯灯色识别系统（不需识别灯向），并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 FPGA 上板实现，仿真系统设计的合理性。

8 基于异构平台的交通信号灯灯向识别系统

设计实现红绿灯灯向识别系统（不需识别灯色），并使用本课程所讲授智能系统优化设计知识，对系统进行软硬件协同设计优化，并进行 FPGA 上板实现，仿真系统设计的合理性。

附录一：gcd.v 文件

gcd.v 文件：

```
//
==========================================================
============
// RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and
SystemC
// Version: 2018.2
// Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
//
//
==========================================================
==========

`timescale 1 ns / 1 ps

(*
CORE_GENERATION_INFO="gcd,hls_ip_2018_2,{HLS_INPUT_TYPE=c,HLS_INPU
T_FLOAT=0,HLS_INPUT_FIXED=1,HLS_INPUT_PART=xc7z020clg484-1,HLS_INP
UT_CLOCK=10.000000,HLS_INPUT_ARCH=others,HLS_SYN_CLOCK=3.412000,
HLS_SYN_LAT=-1,HLS_SYN_TPT=none,HLS_SYN_MEM=0,HLS_SYN_DSP=0,HLS
_SYN_FF=135,HLS_SYN_LUT=132,HLS_VERSION=2018_2}" *)

module gcd (ap_clk,ap_rst,ap_start,ap_done,ap_idle,ap_ready,m,n,ap_return);

parameter    ap_ST_fsm_state1 = 13'd1;
parameter    ap_ST_fsm_state2 = 13'd2;
parameter    ap_ST_fsm_state3 = 13'd4;
parameter    ap_ST_fsm_state4 = 13'd8;
parameter    ap_ST_fsm_state5 = 13'd16;
parameter    ap_ST_fsm_state6 = 13'd32;
parameter    ap_ST_fsm_state7 = 13'd64;
parameter    ap_ST_fsm_state8 = 13'd128;
parameter    ap_ST_fsm_state9 = 13'd256;
parameter    ap_ST_fsm_state10 = 13'd512;
parameter    ap_ST_fsm_state11 = 13'd1024;
parameter    ap_ST_fsm_state12 = 13'd2048;
parameter    ap_ST_fsm_state13 = 13'd4096;

input    ap_clk;
input    ap_rst;
input    ap_start;
```

```
output    ap_done;
output    ap_idle;
output    ap_ready;
input   [7:0] m;
input   [7:0] n;
output   [7:0] ap_return;

reg ap_done;
reg ap_idle;
reg ap_ready;

(* fsm_encoding = "none" *) reg    [12:0] ap_CS_fsm;
wire      ap_CS_fsm_state1;
wire    [7:0] grp_fu_53_p2;
wire      ap_CS_fsm_state13;
reg    [7:0] m_assign_reg_26;
reg    [7:0] p_0_reg_36;
wire      ap_CS_fsm_state2;
wire    [0:0] tmp_fu_47_p2;
reg     grp_fu_53_ap_start;
wire      grp_fu_53_ap_done;
reg    [12:0] ap_NS_fsm;

// power-on initialization
initial begin
#0 ap_CS_fsm = 13'd1;
end

gcd_urem_8ns_8ns_bkb #(
    .ID( 1 ),
    .NUM_STAGE( 12 ),
    .din0_WIDTH( 8 ),
    .din1_WIDTH( 8 ),
    .dout_WIDTH( 8 ))
gcd_urem_8ns_8ns_bkb_U1(
    .clk(ap_clk),
    .reset(ap_rst),
    .start(grp_fu_53_ap_start),
    .done(grp_fu_53_ap_done),
    .din0(p_0_reg_36),
    .din1(m_assign_reg_26),
    .ce(1'b1),
    .dout(grp_fu_53_p2)
);
```

```
always @ (posedge ap_clk) begin
    if (ap_rst == 1'b1) begin
        ap_CS_fsm <= ap_ST_fsm_state1;
    end else begin
        ap_CS_fsm <= ap_NS_fsm;
    end
end

always @ (posedge ap_clk) begin
    if ((1'b1 == ap_CS_fsm_state13)) begin
        m_assign_reg_26 <= grp_fu_53_p2;
    end else if (((1'b1 == ap_CS_fsm_state1) & (ap_start == 1'b1))) begin
        m_assign_reg_26 <= n;
    end
end

always @ (posedge ap_clk) begin
    if ((1'b1 == ap_CS_fsm_state13)) begin
        p_0_reg_36 <= m_assign_reg_26;
    end else if (((1'b1 == ap_CS_fsm_state1) & (ap_start == 1'b1))) begin
        p_0_reg_36 <= m;
    end
end

always @ (*) begin
    if (((tmp_fu_47_p2 == 1'd1) & (1'b1 == ap_CS_fsm_state2))) begin
        ap_done = 1'b1;
    end else begin
        ap_done = 1'b0;
    end
end

always @ (*) begin
    if (((ap_start == 1'b0) & (1'b1 == ap_CS_fsm_state1))) begin
        ap_idle = 1'b1;
    end else begin
        ap_idle = 1'b0;
    end
end

always @ (*) begin
    if (((tmp_fu_47_p2 == 1'd1) & (1'b1 == ap_CS_fsm_state2))) begin
        ap_ready = 1'b1;
```

```
    end else begin
        ap_ready = 1'b0;
    end
end

always @ (*) begin
    if (((tmp_fu_47_p2 == 1'd0) & (1'b1 == ap_CS_fsm_state2))) begin
        grp_fu_53_ap_start = 1'b1;
    end else begin
        grp_fu_53_ap_start = 1'b0;
    end
end

always @ (*) begin
    case (ap_CS_fsm)
        ap_ST_fsm_state1 : begin
            if (((1'b1 == ap_CS_fsm_state1) & (ap_start == 1'b1))) begin
                ap_NS_fsm = ap_ST_fsm_state2;
            end else begin
                ap_NS_fsm = ap_ST_fsm_state1;
            end
        end
        ap_ST_fsm_state2 : begin
            if (((tmp_fu_47_p2 == 1'd1) & (1'b1 == ap_CS_fsm_state2))) begin
                ap_NS_fsm = ap_ST_fsm_state1;
            end else begin
                ap_NS_fsm = ap_ST_fsm_state3;
            end
        end
        ap_ST_fsm_state3 : begin
            ap_NS_fsm = ap_ST_fsm_state4;
        end
        ap_ST_fsm_state4 : begin
            ap_NS_fsm = ap_ST_fsm_state5;
        end
        ap_ST_fsm_state5 : begin
            ap_NS_fsm = ap_ST_fsm_state6;
        end
        ap_ST_fsm_state6 : begin
            ap_NS_fsm = ap_ST_fsm_state7;
        end
        ap_ST_fsm_state7 : begin
            ap_NS_fsm = ap_ST_fsm_state8;
        end
```

```
        ap_ST_fsm_state8 : begin
            ap_NS_fsm = ap_ST_fsm_state9;
        end
        ap_ST_fsm_state9 : begin
            ap_NS_fsm = ap_ST_fsm_state10;
        end
        ap_ST_fsm_state10 : begin
            ap_NS_fsm = ap_ST_fsm_state11;
        end
        ap_ST_fsm_state11 : begin
            ap_NS_fsm = ap_ST_fsm_state12;
        end
        ap_ST_fsm_state12 : begin
            ap_NS_fsm = ap_ST_fsm_state13;
        end
        ap_ST_fsm_state13 : begin
            ap_NS_fsm = ap_ST_fsm_state2;
        end
        default : begin
            ap_NS_fsm = 'bx;
        end
    endcase
end

assign ap_CS_fsm_state1 = ap_CS_fsm[32'd0];

assign ap_CS_fsm_state13 = ap_CS_fsm[32'd12];

assign ap_CS_fsm_state2 = ap_CS_fsm[32'd1];

assign ap_return = p_0_reg_36;

assign tmp_fu_47_p2 = ((m_assign_reg_26 == 8'd0) ? 1'b1 : 1'b0);

endmodule //gcd
```

附录二：gcd_urem_8ns_8ns_bkb.v 文件

文件：gcd_urem_8ns_8ns_bkb.v

```
//
==========================================================
============
// File generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and
SystemC
// Version: 2018.2
// Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
//
//
==========================================================
============

`timescale 1 ns / 1 ps

module gcd_urem_8ns_8ns_bkb_div_u
#(parameter
    in0_WIDTH = 32,
    in1_WIDTH = 32,
    out_WIDTH = 32
)
(
    input                           clk,
    input                           reset,
    input                           ce,
    input                           start,
    input        [in0_WIDTH-1:0] dividend,
    input        [in1_WIDTH-1:0] divisor,
    output wire                     done,
    output wire [out_WIDTH-1:0] quot,
    output wire [out_WIDTH-1:0] remd
);

localparam cal_WIDTH = (in0_WIDTH > in1_WIDTH)? in0_WIDTH : in1_WIDTH;

//------------------------Local signal-------------------
reg      [in0_WIDTH-1:0] dividend0;
reg      [in1_WIDTH-1:0] divisor0;
reg      [in0_WIDTH-1:0] dividend_tmp;
reg      [in0_WIDTH-1:0] remd_tmp;
```

```
wire     [in0_WIDTH-1:0] dividend_tmp_mux;
wire     [in0_WIDTH-1:0] remd_tmp_mux;
wire     [in0_WIDTH-1:0] comb_tmp;
wire     [cal_WIDTH:0]    cal_tmp;

//-----------------------Body--------------------------
assign   quot    = dividend_tmp;
assign   remd    = remd_tmp;

// dividend0, divisor0
always @(posedge clk)
begin
    if (start) begin
        dividend0 <= dividend;
        divisor0  <= divisor;
    end
end

// One-Hot Register
// r_stage[0]=1:accept input; r_stage[in0_WIDTH]=1:done
reg      [in0_WIDTH:0]      r_stage;
assign done = r_stage[in0_WIDTH];
always @(posedge clk)
begin
    if (reset == 1'b1)
        r_stage[in0_WIDTH:0] <= {in0_WIDTH{1'b0}};
    else if (ce)
        r_stage[in0_WIDTH:0] <= {r_stage[in0_WIDTH-1:0], start};
end

// MUXs
assign   dividend_tmp_mux = r_stage[0]? dividend0 : dividend_tmp;
assign   remd_tmp_mux      = r_stage[0]? {in0_WIDTH{1'b0}} : remd_tmp;

if (in0_WIDTH == 1) assign comb_tmp = dividend_tmp_mux[0];
else                  assign comb_tmp = {remd_tmp_mux[in0_WIDTH-2:0],
dividend_tmp_mux[in0_WIDTH-1]};

assign   cal_tmp   = {1'b0, comb_tmp} - {1'b0, divisor0};

always @(posedge clk)
begin
    if (ce) begin
        if (in0_WIDTH == 1) dividend_tmp <= ~cal_tmp[cal_WIDTH];
```

```
        else            dividend_tmp <= {dividend_tmp_mux[in0_WIDTH-2:0],
~cal_tmp[cal_WIDTH]};
        remd_tmp      <= cal_tmp[cal_WIDTH]? comb_tmp :
cal_tmp[in0_WIDTH-1:0];
    end
end

endmodule

module gcd_urem_8ns_8ns_bkb_div
#(parameter
        in0_WIDTH    = 32,
        in1_WIDTH    = 32,
        out_WIDTH    = 32
)
(
        input                           clk,
        input                           reset,
        input                           ce,
        input                           start,
        output  reg                      done,
        input           [in0_WIDTH-1:0] dividend,
        input           [in1_WIDTH-1:0] divisor,
        output  reg     [out_WIDTH-1:0] quot,
        output  reg     [out_WIDTH-1:0] remd
);
//------------------------Local signal-------------------
reg                       start0 = 'b0;
wire                      done0;
reg     [in0_WIDTH-1:0] dividend0;
reg     [in1_WIDTH-1:0] divisor0;
wire    [in0_WIDTH-1:0] dividend_u;
wire    [in1_WIDTH-1:0] divisor_u;
wire    [out_WIDTH-1:0] quot_u;
wire    [out_WIDTH-1:0] remd_u;
//------------------------Instantiation------------------
gcd_urem_8ns_8ns_bkb_div_u #(
    .in0_WIDTH       ( in0_WIDTH ),
    .in1_WIDTH       ( in1_WIDTH ),
    .out_WIDTH       ( out_WIDTH )
) gcd_urem_8ns_8ns_bkb_div_u_0 (
    .clk        ( clk ),
    .reset      ( reset ),
    .ce         ( ce ),
```

```
    .start     ( start0 ),
    .done      ( done0 ),
    .dividend ( dividend_u ),
    .divisor   ( divisor_u ),
    .quot      ( quot_u ),
    .remd      ( remd_u )
);
//-----------------------Body--------------------------
assign dividend_u = dividend0;
assign divisor_u = divisor0;

always @(posedge clk)
begin
    if (ce) begin
        dividend0 <= dividend;
        divisor0  <= divisor;
        start0    <= start;
    end
end

always @(posedge clk)
begin
    done <= done0;
end

always @(posedge clk)
begin
    if (done0) begin
        quot <= quot_u;
        remd <= remd_u;
    end
end

endmodule



`timescale 1 ns / 1 ps
module gcd_urem_8ns_8ns_bkb(
    clk,
    reset,
    ce,
    start,
    done,
```

```
    din0,
    din1,
    dout);

parameter ID = 32'd1;
parameter NUM_STAGE = 32'd1;
parameter din0_WIDTH = 32'd1;
parameter din1_WIDTH = 32'd1;
parameter dout_WIDTH = 32'd1;
input clk;
input reset;
input ce;
input start;
output done;
input[din0_WIDTH - 1:0] din0;
input[din1_WIDTH - 1:0] din1;
output[dout_WIDTH - 1:0] dout;

wire[dout_WIDTH - 1:0] sig_quot;


gcd_urem_8ns_8ns_bkb_div #(
.in0_WIDTH( din0_WIDTH ),
.in1_WIDTH( din1_WIDTH ),
.out_WIDTH( dout_WIDTH ))
gcd_urem_8ns_8ns_bkb_div_U(
    .dividend( din0 ),
    .divisor( din1 ),
    .remd( dout ),
    .quot( sig_quot ),
    .clk( clk ),
    .ce( ce ),
    .reset( reset ),
    .start( start ),
    .done( done ));

endmodule
```

附录三：TRS.py 文件

TRS.py

```
from pynq import Overlay , PL #提供的包，用于调用硬件核 PYNQIP
from PIL import Image #用于图片处理
import numpy as np
import cffi #用于调用 C++代码
import os
import tempfile

BNN_ROOT_DIR = os.path.dirname(os.path.realpath(__file__))
BNN_LIB_DIR = os.path.join(BNN_ROOT_DIR , 'libraries ')
BNN_BIT_DIR = os.path.join(BNN_ROOT_DIR , 'bitstreams ')
BNN_PARAM_DIR = os.path.join(BNN_ROOT_DIR , 'params ')

RUNTIME_HW = "python_hw"
RUNTIME_SW = "python_sw"
_ffi = cffi.FFI()

_ffi.cdef("""
void load_parameters(const char* path);
unsigned int inference(const char* path, unsigned int results[64], int number_class ,
float *usecPerImage);
unsigned int* inference_multiple(const char* path , int number_class ,
                    int * image_number , float * usecPerImage ,     unsigned int
                    enable_detail);
void free_results(unsigned int * result);
void deinit ();
"""
)

_libraries = {}

class TSR:
def __init__(self , runtime=RUNTIME_HW , load_overlay=True):
    self.bitstream_name = None
   if runtime == RUNTIME_HW:
      self.bitstream_name="cnv -pynq -pynq.bit".format(network)
     self.bitstream_path=os.path.join(BNN_BIT_DIR , self.bitstream_name)
   if PL.bitfile_name != self.bitstream_path:
      if load_overlay:
```

```
36          #调用比特流文件
37          Overlay(self.bitstream_path).download ()
38      else :
39          raise RuntimeError("Incorrect Overlay loaded")
40    #调用 C++
41    dllname = "{0}-cnv -pynq.so".format(runtime, network)
42    if dllname not in _libraries:
43      _libraries[dllname] = _ffi.dlopen( os.path.join(BNN_LIB_DIR, dllname))
45    self.interface = _libraries[dllname]
46    self.num_classes = 0
47
48  def __del__(self):
49      self.interface.deinit ()
50
51  #加载参数 BNN
52  def load_parameters(self):
53      if not os.path.isabs("road -signs"):
54          params = os.path.join(BNN_PARAM_DIR, "road -signs")
55      self.interface.load_parameters(params.encode ())
56      self.classes = []
57      with open (os.path.join(params, "classes.txt")) as f:
58          self.classes = [c.strip () for c in f.readlines ()]
59      filter(None , self.classes)
60
61  #识别标志
62  def inference_multiple(self, path):
63      size_ptr = _ffi.new("int *")
64      usecperimage = _ffi.new("float *")
65      result_ptr = self.interface.inference_multiple(
66          path.encode (), len(self.classes), size_ptr,usecperimage,0)
67    result_buffer = _ffi.buffer(result_ptr, size_ptr[0] * 4)
68    result_array = np.copy(np.frombuffer(result_buffer,dtype=np.uint32))
69    self.interface.free_results(result_ptr)
70    time = usecperimage[0]*size_ptr[0]
71    result_array = result_array.tolist ().insert(0, time)
72    return result_array
73
74   def class_name(self, index):
75      return self.classes[index]
76
77 #分类器 BNN
78 class Classifier:
79      def __init__(self, runtime=RUNTIME_HW):
80          self.tsr = TSR(runtime)
```

```
81      self.tsr.load_parameters ()
82
83 #图像预处理
84   def image_to_cifar(self, im , fp):
85      im.thumbnail ((32, 32), Image.ANTIALIAS)
86      background = Image.new('RGBA', (32 , 32), (255 , 255 , 255 ,0))
87     background.paste( im , (int((32 - im.size[0]) / 2), int((32 - im.size[1]) / 2))
89        )
90       im = (np.array(background))
91       r = im[:,:,0].flatten ()
92       g = im[:,:,1].flatten ()
93       b = im[:,:,2].flatten ()
94       label = np.identity(1, dtype=np.uint8)
95       fp.write(label.tobytes ())
96       fp.write(r.tobytes ())
97       fp.write(g.tobytes ())
98       fp.write(b.tobytes ())
99
100 #图像识别
101   def classify_images(self, ims):
102        with tempfile.NamedTemporaryFile () as tmp:
103            for im in ims:
104                self.image_to_cifar(im , tmp)
105            tmp.flush ()
106            return self.tsr.inference_multiple(tmp.name)
107
108    def class_name(self, index):
109       return self.tsr.classes[index]
```