华 中 科 技 大 学 课 程 设 计 报 告

数字电路与逻辑设计

课程设计报告

团队成员姓名	班级	学号	贡献百分比
李真	ACM1501	U201514527	100%

课程设计实验部分：实验完成情况、时间 （亮点、完成、部分完成）		总分 （实验部分 70% +报告 30%）
	设计	
检查结果		
检查名次		
检查老师		

报告人　　：李　真

实验指导教师：何云峰

报告批阅教师：

计算机科学与技术学院

2017 年 6 月 13 日

华 中 科 技 大 学 课 程 设 计 报 告

数字逻辑课程设计学生工作表

班 级	姓 名	学 号	验收时间（教师填写）
ACM1501	李真	U201514527	

（学生填写）

课程设计具体工作

1、姓名：李真

设计：洗衣机控制器

负责：分析需求、确定自定义需求、确定系统 IO、确定所需模块、确定代码接口、确定代码规范、实现各模块、进行仿真调试、进行功能测试。

2、亮点

（1）新功能：儿童锁。

（2）新功能：舱门锁。

（3）改动：电源改为按钮，添加防误关机及强制断电功能。

（4）改动：因为元件短缺，水位由自动检测改为手动选择。

（5）设计：综合设计和实现设计没有任何错误、严重警告或警告。

课设进度记录（学生填写）

日期	进度
6.5	接受任务； 进一步学习 Verilog HDL 的设计思想、方法； 实现部分底层模块。
6.6	继续学习 Verilog HDL 的设计思想、方法； 实现大部分功能模块并进行仿真； 开始构思附加功能。
6.7	实现所有基本功能，进行仿真； 继续思考附加功能。
6.8	进行功能测试； 修改代码，改善设计； 继续思考附加功能。
6.9	实现附加功能； 整理代码，代码风格统一化； 进行完整功能测试。
6.12	开始编写报告。
6.13	完成报告。

实验平台故障记录（学生填写，请注明实验平台的编号）

（无）

华 中 科 技 大 学 课 程 设 计 报 告

重要说明

1、 **时间安排：课内** 2 周。

2、 验收准备：

1） 完成本表学生应该填写部分；

2） 同组的每位学生必须都能**以独自完成的方式**应对任何形式的验收；

3） 完成课程设计报告书（**格式参见模板）**；

4） 将源程序和报告的电子文档交班长。

3、 检查过程：

1） 提交验收准备材料，请求老师验收，之后按验收老师的要求做；

2） 在开发平台上根据验收老师的要求进行演示；

3） 检查过程中独立回答老师提出的相关问题；

4） 验收老师有权根据具体情况调整验收的内容与方式；

5） 验收完成后关闭电源，整理好设备。

4、 评分标准：

1) 同组者工作量的分配；

2) 在完成控制器基本要求外，有**亮点**为加分项；

3) 在规定时间内完成控制器基本要求；

4) 在规定时间内完成控制器**部分**基本要求；

5) 检查时间。

5、 课程设计判定为不合格的一些情形： （本人已阅读此条款 1-5 项：签名_____）

1） 请人代做或冒名顶替者；

2） 替人做且不听劝告者；

3） 课程设计报告内容抄袭或雷同者；

4） 课程设计报告内容与实际实验内容不一致者；

5） 课程设计代码抄袭者。

目　录

1 课程设计概述

1.1 课设目的

（1）掌握 Vivado 软件的使用方法；

（2）熟悉 FPGA 器件的使用方法；

（3）用 Verilog HDL 进行较复杂逻辑电路的设计和调试；

（4）学习数字系统的设计方法；

（5）通过规范化的实验报告，培养学生良好的文档习惯以及撰写规范文档的能力。

1.2 课设要求

（1）能够全面地应用课程中所学的基本理论和基本方法，完成从设计逻辑电路到设计简单数字系统的过渡；

（2）能力独立思考、独立查阅资料，独立设计规定的系统；

（3）能够独立地完成实施过程，包括电路设计、调试、排除故障、仿真和下载验证。

1.3 课设任务

具体参见数字逻辑课程设计题目。

（1）通过 Verilog HDL 完成规定的设计任务，采取模块化、层次化的设计方法设计电路，然后进行编译和仿真，认真记录实施过程中遇到的故障以及解决方法，保证设计的正确性；

（2）生成 bit 文件，下载到开发板上，通过实际线路进行验证设计的正确性；

（3）撰写设计报告，并对存在的问题进行分析、提出改进意见。

1.4 实验环境

开发环境为 Vivado 2017.1 软件和开发板 Nexys4-DDR（芯片为 XC7A100TCSG324-1，封装为

CSG3242）。 Vivado 2017.1 是使用 Xilinx FPGA 必备的设计工具。它可以完成 FPGA 开发的全部流程，包括设计输入、仿真、 综合、布局布线、生成 bit 文件、配置以及在线调试等功能。

Nexys4-DDR 开发板简介：参见图 1-1 所示，它是一款简单易用的数字电路开发平台，可以支持在课堂环境中来设计一些行业应用。大规模、高容量的 FPGA，海量的外部存储，各种 USB、以太网、以及其它接口、这些让 Nexys4-DDR 能够满足从入门级组合逻辑电路到强大的嵌入式系统的设计。同时，板上集成的加速度、温度传感器，MEMs 数字麦克风，扬声器放大器以及人量的 I/O 设备，让 Nexys4-DDR 不需要增添额外组件而用于各种各样的设计。

注意：开发板提供的时钟信号频率为 100 MHz，对应的引脚封装编号为“E3”。

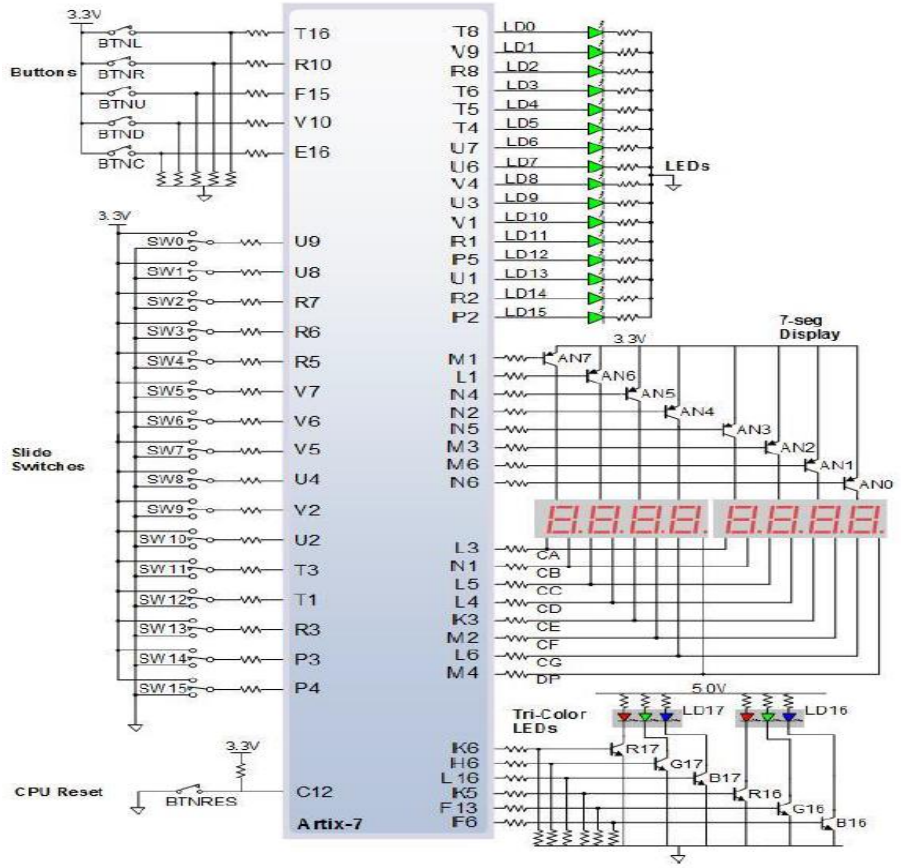

图 1-1 Nexys4-DDR 通用 I/O 设备

2 洗衣机控制系统设计

2.1 目的

（1）掌握较复杂的逻辑设计和调试

（2）学习用原理图+Verilog HDL 语言设计逻辑电路；

（3）学习数字电路模块、层次设计；

（4）掌握 Vivado 软件的使用方法；

（5）熟悉 FPGA 器件的使用。

2.2 内容

设计并利用 Nexys4 开发板来实现一个全自动洗衣机控制系统，具体要求如下。

（1）电源开关同时作为电路总清零信号（Reset），当 Reset = Off 不工作、为 On 时电路进入初始状态，且电源指示灯亮（又称洗衣机控制器运行指示灯），控制器初始状态为：“洗漂脱”。

（2）启动

在电源打开，“启动”指示灯不亮的情况下，按一下“启动/暂停”，“启动”指示灯亮，洗衣机开始工作或恢复暂停时的工作状态并开始工作。

（3）暂停

在电源打开，“启动”指示灯亮的情况下，按一下“启动/暂停”，“启动”指示灯灭，此时洗衣机处于“暂停”状态，等待重新“启动”。暂停后可以重新选择洗衣程序。

（4）洗衣机洗衣程序

a. 时间设置

进水为 3 分钟（为了方便描述假定衣物重量为 3kg）；洗衣 9 分钟； 排水 3 分钟；脱水/甩干 3 分钟，漂洗 6 分钟。

说明：进水时间随衣物重量改变，这里假定衣物重量分为 5kg、4kg、3kg、2kg 4 种情况，控制器的“水位”控制分别对应为进水 5、4、3、2 分钟。

b. 洗涤程序

进水---洗衣，共 12 分钟；正在进行洗涤时，洗涤灯闪烁，正在进水时，进水灯亮。

c. 漂洗程序

排水---甩干---进水---漂衣，共 15 分钟，正在进行漂洗时，漂洗灯闪烁，正在进水时，进水灯亮，正在进行排水和脱水/甩干时，对应指示灯亮。

d. 脱水程序

排水---甩干，共 6 分钟。正在进行排水时，排水灯闪烁，正在进行脱水/甩干时，脱水灯亮。

（5）洗衣模式：目前考虑有以下 6 种洗衣模式可供选择。

为了方便提示洗衣过程，假定未完成的洗衣过程其对应的指示灯亮，正在进行的洗衣过程其对应的指示灯闪烁，已完成的洗衣过程前对应指示灯熄灭。

a. 洗漂脱

它含有“洗涤”、“漂洗”、“脱水”3 个过程，整个程序流程包括：洗涤 12 分钟（进水---洗衣）---漂洗 15 分钟（排水---甩干---进水---漂洗）----脱水 6 分钟（排水---甩干），共 33 分钟。

b. 单洗（仅洗涤）

它含有洗涤 1 个过程，整个程序流程包括：洗涤 12 分钟（进水---洗衣）。

c. 洗漂

它含有“洗涤”、“漂洗”2 个过程。整个程序流程包括：洗涤 12 分钟（进水---洗衣）---漂洗 15 分钟（排水---甩干---进水---漂衣），共 27 分钟；

d. 单漂（仅漂洗）

它含有“漂洗”1 个过程。整个程序流程包含：漂洗 15 分钟（排水---甩干---进水---漂衣）。

e. 漂脱

它含有“漂洗”、“脱水”2 个过程。整个程序流程包含：漂洗 15 分钟（排水---甩干---进水---漂衣）----脱水 6 分钟（排水---甩干），共 21 分钟。

f. 单脱（仅脱水）

它含有“脱水”1 个过程。整个程序流程包含：脱水 6 分钟（排水---甩干）。

（6）模式选择

系统共有 6 种洗衣模式供选择，初始状态为“洗漂脱”洗衣模式，每按一次模式选择，洗衣模式改变一次，改变顺序为：“洗漂脱”----“单洗”----“洗漂”----“单漂”----“漂脱”----“单脱”----“洗漂脱”循环。

在选择洗衣模式时，总剩余时间 7 段数码管会显示新的洗衣模式的总时间，当前模式时间 7 段数码显示管会显示当前所选洗衣模式中第一个洗衣程序的所需时间。同时“洗涤”、“漂洗”和“脱水”指示灯也随着新的洗衣模式的变化而变化。某个指示灯亮，表示在本洗衣模式中包含对应的工作流程；反之则不包含。

洗衣模式选择完毕，按启动按钮，该洗衣模式启动，当该洗衣模式结束时要发出报警声，并回到洗漂脱模式；

（7）水位

控制器会根据衣物重量（假定衣物重量分为 5kg、4kg、3kg、2kg 4 种情况）自动选择“水位”，水位由 2 个 7 段数码显示管指示；

（8）蜂鸣

在按每个按钮时，都要有“嘟”的一声提示，在每个程序结束时，应有“嘟嘟嘟”“嘟嘟嘟”，“嘟嘟嘟”三声提示，同时“启动”指示灯熄灭。（由于器件短缺，用蜂鸣指示灯的闪烁表示报警效果）

（9）关闭

每次所选的洗衣模式结束全部工作后，10 秒内无操作，自动执行“电源”开关关闭操作。

（10）自定义规则

a. 电源开关：根据原设计内容，为了方便实现自动关机，采用按钮形式的开关。

b. 防误关机：为防止在洗衣过程中意外关闭洗衣机，在洗衣或暂停时不响应关机信号。

c. 强制断电：考虑到在意外情况发生时需要强制断电，在电源打开的状态下，按下电源按钮不放的两秒后强制关闭洗衣机。（此操作相当于于原内容要求的 Reset 信号）

d. 舱门锁：为防止在洗衣过程中意外开启洗衣机舱门，在洗衣或暂停时使舱门不可开启。（由于器件短缺，用舱门锁指示灯表示舱门锁效果）

e. 暂停表示：为了方便表示洗衣过程已暂停，在暂停状态时数码管显示由常亮改为闪烁，非暂停状态为常亮（电源开）。

f. 水位选择：因为元件短缺，所以水位不采用自动检测，而改用手动选择。共有 6 种水位选择，初始值对应 3kg 衣量，每按一次水位选择水位改变一次，其对应衣量的改变顺序为：3kg ---- 4kg ---- 5kg ---- 2kg ---- 3kg 循环。每个洗衣过程所需时间与水位成正比。

g. 儿童锁：为防止来自小儿或外人的误操作，可以在同时按下模式选择和水位选择的一秒后打开儿童锁，此时儿童锁指示灯亮。在儿童锁打开状态下，同时按下模式选择和水位选择的一秒后可以关闭儿童锁，此时儿童锁指示灯灭。在儿童锁打开状态下，除了强制断电和关闭儿童锁外，任何输入都不被响应。

h. 按钮响应：综合原设计内容和上述自定义规则，除开关儿童锁和强制断电外，系统统一在按钮松开时响应。开关儿童锁和强制断电均在按下相应按钮并等待所需时间后进行响应。

i. 时间缩短：考虑到原规则要求的时间都是以分钟为单位，验收会比较耗时。为了方便验收，洗衣耗时均改为以秒为单位（即原时间要求的 1/60），自动关机的时间还是 10 秒不变。

2.3 设计思路

人机交互界面如图 2-1 所示。

图 2-1 洗衣机控制器人机交互界面

系统结构如图 2-2 所示，接受来自四个按钮和一个系统时钟的输入，控制数码管和各指示灯及蜂鸣器和舱门锁的输出。

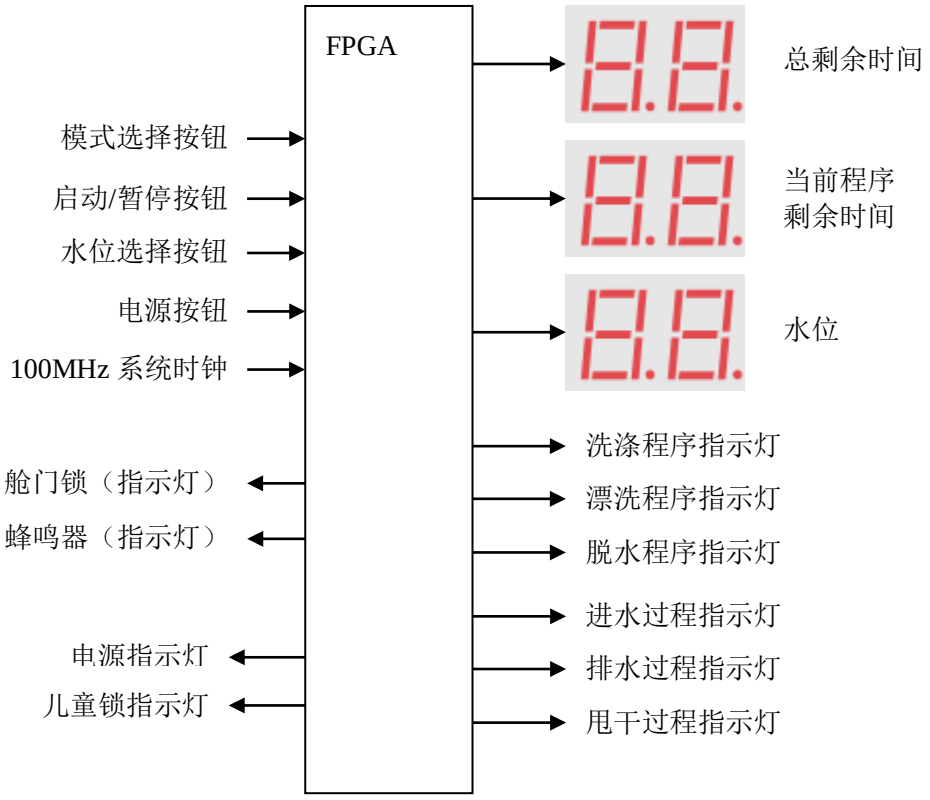

图 2-2 洗衣机控制器系统结构框图

（1）模块化

使用模块化设计，主要模块如下共 15 类：

顶层（top）、电源（power）、儿童锁（protector）、按钮（button）、防抖动（debouncer）、蜂鸣器（buzzer）、LED（leds）、数码管（display）、主状态机（main）、设置（setting）、模式运行（run_mode）、程序运行（run_prog）、嵌套定时器（timer_nested）、定时器（timer）、分频器（divider）。

（2）系统 IO

由顶层模块负责，输入端口如表 2-1 所示，输出端口如表 2-2 所示。

表 2-1 顶层模块输入端口

端口名称	端口含义	开发板对应器件	开发板对应引脚
a_mod	模式选择按钮	按钮 BTNL	P17
a_run	启动/暂停按钮	按钮 BTNC	N17
a_wat	水位选择按钮	按钮 BTNR	M17
a_pwr	电源按钮	按钮 BTNU	M18
clk	100 MHz 系统时钟	100 MHz 晶振	E3

表 2-2 顶层模块输出端口

端口名称	端口含义	开发板对应器件	开发板对应引脚
buz	蜂鸣器（使用 LED 模拟）	发光二极管 LD15	V11
ioh	舱门锁	发光二极管 LD14	V12
led_pwr	电源指示灯	发光二极管 LD0	H17
led_lck	儿童锁指示灯	发光二极管 LD1	K15
led_dry	脱水程序指示灯	发光二极管 LD7	U16
led_rin	漂洗程序指示灯	发光二极管 LD8	V16
led_was	洗涤程序指示灯	发光二极管 LD9	T15
led_fil	进水过程指示灯	发光二极管 LD3	N14
led_spi	甩干过程指示灯	发光二极管 LD4	R18
led_dra	排水过程指示灯	发光二极管 LD5	V17
seg_n	数码管段选择位（8 位，低电平有效）	数码管 DISP1 和 DISP2	（见注 1）
an_n	数码管位使能位（8 位，低电平有效）	数码管 DISP1 和 DISP2	（见注 2）

注 1：seg_n 的引脚为：T10，R10，K16，K13，P15，T11，L18，H15。

注 2：an_n 的引脚为：J17，J18，T9，J14，P14，T14，K2，U13。

（3）总体规则

a. 除系统时钟外所有输入信号全部同步化、防抖动处理。

b. 脉冲信号长度统一为一个系统时钟周期。

c. 所有模块均接受系统时钟输入。

d. 除电源外所有模块均接受电源信号输入。电源信号同时为除电源模块以外所有模块的复位信号（低电平有效）。

e. 顶层模块中实例化一个提供 1Hz 时钟的分频器，用于控制数码管和 LED 的闪烁。

其他延时执行的任务由各模块自行实例化定时器并使用。

f. 各任务模块（主状态机、设置、模式运行、程序运行、定时器）在其任务执行完成后都应发出完成信号，以告诉上层模块进入下一步处理。

g. 主要状态编码

主状态机：使用 2 位二进制数依次表示：设置、运行、等待关机、暂停状态。

模式编码：使用 3 位掩码来表示模式所需程序，运行时从低位到高位执行对应程序。

过程编码：使用 5 位掩码来表示程序所需流程，运行时从低位到高位执行对应过程。

（4）各模块功能描述

a. 顶层模块（top）

功能：负责将各系统 IO 端口连接至各主要功能模块，以及各主要功能模块之间的连接。

b. 电源模块（power）

功能：控制整个系统的电源、复位。

输入信号：电源按钮，主状态机的完成信号，儿童锁状态。

输出脉冲：电源触发脉冲（电源按钮信号的下降沿脉冲）。

输出状态：电源状态，电源指示灯。

c. 儿童锁模块（protector）

功能：决定儿童锁状态。

输入信号：模式选择按钮，水位选择按钮。

输出脉冲：儿童锁触发脉冲（同时按住模式选择和水位选择后 1 秒发出该脉冲信号）。

输出状态：儿童锁状态，儿童锁指示灯。

d. 按钮模块（button）

功能：使用防抖动模块将对应按钮输入信号进行同步化、防抖动处理，并根据儿童锁状态给

出按钮的触发脉冲。

输入信号：对应按钮，儿童锁状态。

输出脉冲：对应按钮的触发脉冲（按钮信号的下降沿脉冲）。

e. 防抖动模块（debouncer）

功能：对异步输入信号进行同步化、防抖动处理。

输入信号：异步信号。

输出脉冲：上升沿脉冲，下降沿脉冲。

输出状态：同步化后的信号状态。

f. 蜂鸣器模块（buzzer）

功能：根据各触发信号和程序完成信号控制蜂鸣器。“嘟”提示（用于各触发信号）蜂鸣长度为 0.1 秒；“嘟嘟嘟，嘟嘟嘟，嘟嘟嘟”提示（用于程序完成信号）蜂鸣长度为 0.1 秒，同组相邻两声“嘟”间隔 0.1 秒，相邻两组“嘟嘟嘟”间隔 0.3 秒。

输入信号：各触发脉冲。

输出状态：蜂鸣器。

g. LED 模块（leds）

功能：控制用于表示洗衣状态的指示灯（各程序指示灯和各过程指示灯）。

输入信号：各程序指示灯的常亮、闪烁状态，各过程指示灯的常亮状态，1Hz 时钟。

输出信号：各程序指示灯，各过程指示灯。

h. 数码管（display）

功能：控制数码管显示。第 7、6 位显示总剩余时间，第 5、4 位显示当前模式剩余时间，第 3、2 位常灭，第 1、0 位显示水位。

输入信号：总剩余时间，当前模式剩余时间，水位，闪烁状态，1 Hz 时钟。

输出状态：数码管的段选择位和使能位。

i. 主状态机模块（main）

功能：描述系统主体功能行为。

输入信号：电源、模式选择、启动/暂停、水位选择按钮的触发信号。

输出脉冲：完成脉冲，程序完成脉冲。

输出状态：舱门锁状态，各程序指示灯的常亮、闪烁状态，各过程指示灯的常亮状态，总剩余时间，当前模式剩余时间，水位，数码管闪烁状态。

j. 设置模块（setting）

功能：描述在设置洗衣模式、水位时的功能行为。

输入信号：模式选择、启动/暂停、水位选择按钮的触发信号，使能状态。

输出脉冲：完成脉冲。

输出状态：所选模式。

k. 模式运行（run_mode）

功能：描述运行时由洗衣模式决定的功能行为。

输入信号：水位，所需程序，暂停状态，使能状态。

输出脉冲：完成脉冲，程序完成脉冲。

输出状态：各程序指示灯的常亮、闪烁状态，各过程指示灯的常亮状态，总剩余时间，当前模式剩余时间。

l. 程序运行（run_prog）

功能：描述运行时由洗衣过程决定的功能状态。

输入信号：水位，所需过程，使能状态。

输出脉冲：完成脉冲。

输出状态：各过程指示灯的常亮状态，当前程序剩余时间。

m. 嵌套定时器（timer_nested）

功能：在指定时间后发出完成脉冲并提供实时倒计时（单位：秒）。

输入信号：定时（单位：秒），暂停状态，使能状态。

输出脉冲：完成脉冲。

输出状态：剩余时间（单位：秒）。

n. 定时器（timer）

功能：在指定时间后发出完成脉冲。

输入信号：定时（单位：系统时钟周期），使能状态。

输出脉冲：完成脉冲。

o. 分频器模块（divider）

功能：根据参数获取低频时钟信号。

输出状态：分频后的时钟信号。

2.4 代码实现

顶层模块主要结构如图 2-3 所示。

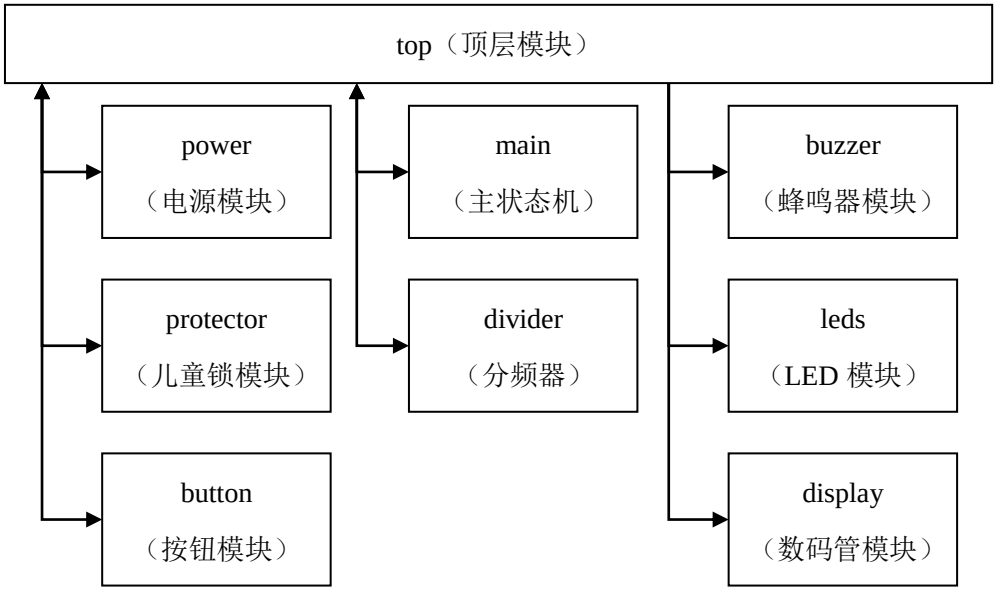

图 2-3 洗衣机控制器顶层原理图

（1）顶层模块 top（Verilog）

程序 2-1：top.v（部分）

```
module top(
    buz, ioh,
    led_pwr, led_lck,
    led_dry, led_rin, led_was,
    led_fil, led_spi, led_dra,
    seg_n, an_n,
    a_mod, a_run, a_wat, a_pwr,
    clk
);
    parameter TIM_UNIT = `c_ms(1000);
    parameter END_WAIT = `c_s(10);
    parameter DEB_WAIT = `c_ms(5);
    parameter LCK_WAIT = `c_ms(1000);
    parameter FPO_WAIT = `c_ms(2000);
    parameter BUZ_INTV = `c_ms(100);
    parameter FLA_INTV = `c_ms(500);
    parameter DIS_INTV = `c_ms(1);
    output buz, ioh; // 蜂鸣器，舱门锁
    output led_pwr, led_lck; // 电源指示灯，儿童锁指示灯
    output led_dry, led_rin, led_was; // 脱水程序指示灯，漂洗程序指示灯，洗涤程序指示灯
    output led_fil, led_spi, led_dra; // 进水过程指示灯，甩干过程指示灯，排水过程指示灯
    output [7:0] seg_n, an_n; // 数码管
    input a_mod, a_run, a_wat, a_pwr; // 模式选择按钮，启动/暂停按钮，水位选择按钮，电源按钮
    input clk; // 100 MHz 系统时钟

    wire a_lck = a_mod && a_wat; // 儿童锁异步输入信号
```

```
wire clk_fl; // 用于 LED 和数码管闪烁的 1 Hz 时钟信号
wire [2:0] ld_drw, fl_drw, ld_fsd; // LED 控制掩码
wire [5:0] u_tot, u_cur, u_wat; // 总时间，当前程序时间，水位
wire fl_disp; // 数码管是否闪烁（洗衣是否暂停）
wire main_done, prog_done; // 主状态机完成，程序完成
wire rst_n, tr_pwr; // 电源信号（复位信号，低电平有效），电源按钮触发脉冲
wire lock; // 儿童锁状态
wire tr_lck, tr_mod, tr_run, tr_wat; // 锁触发脉冲，模式选择、启动/暂停、水位选择触发脉冲
wire buz_tr_a = tr_lck || tr_mod || tr_run || tr_wat; // 蜂鸣（触发脉冲）
wire buz_tr_b = prog_done; // 蜂鸣（程序完成）
// fwd buz
// fwd led_pwr
// fwd led_lck
// fwd led_dry
// fwd led_rin
// fwd led_was
// fwd led_fil
// fwd led_spi
// fwd led_spi
// fwd led_dra
// fwd seg_n
// fwd an_n
// 主状态机
main #(
    .TIM_CMAX(TIM_UNIT),
    .END_CMAX(END_WAIT)
) x_main(
    .done(main_done),
    .prog_done(prog_done),
    .ioh(ioh),
    .ld_drw(ld_drw),
    .fl_drw(fl_drw),
    .ld_fsd(ld_fsd),
    .u_tot(u_tot),
    .u_cur(u_cur),
    .u_wat(u_wat),
    .fl_disp(fl_disp),
    .tr_pwr(tr_pwr),
    .tr_mod(tr_mod),
    .tr_run(tr_run),
    .tr_wat(tr_wat),
    .clk(clk),
    .rst_n(rst_n)
);
// 电源模块
power #(
    .TIM_CMAX(FPO_WAIT),
    .DEB_CMAX(DEB_WAIT)
) x_power(
```

```
        .rst_n(rst_n),
        .tr_pwr(tr_pwr),
        .led_pwr(led_pwr),
        .a_pwr(a_pwr),
        .main_done(main_done),
        .lock(lock),
        .clk(clk)
    );
    // 儿童锁模块
    protector #(
        .TIM_CMAX(LCK_WAIT),
        .DEB_CMAX(DEB_WAIT)
    ) x_protector(
        .lock(lock),
        .tr_lck(tr_lck),
        .led_lck(led_lck),
        .a_lck(a_lck),
        .clk(clk),
        .rst_n(rst_n)
    );
    // 按钮模块
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_mod(
        .tr_btn(tr_mod),
        .a_btn(a_mod),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_run(
        .tr_btn(tr_run),
        .a_btn(a_run),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_wat(
        .tr_btn(tr_wat),
        .a_btn(a_wat),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    // 蜂鸣器模块
    buzzer #(
        .FLA_CMAX(BUZ_INTV)
```

```
    ) x_buzzer(
        .buz(buz),
        .tr_a(buz_tr_a),
        .tr_b(buz_tr_b),
        .clk(clk),
        .rst_n(rst_n)
    );
    // 1 Hz 时钟分频器模块
    divider #(
        .CMAX(FLA_INTV)
    ) x_divider_fl(
        .clk_div(clk_fl),
        .clk(clk),
        .rst_n(rst_n)
    );
    // LED 模块
    leds x_leds(
        .led_dry(led_dry),
        .led_rin(led_rin),
        .led_was(led_was),
        .led_fil(led_fil),
        .led_spi(led_spi),
        .led_dra(led_dra),
        .ld_drw(ld_drw),
        .fl_drw(fl_drw),
        .ld_fsd(ld_fsd),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    //  数码管模块
    display #(
        .SCA_CMAX(DIS_INTV)
    ) x_display(
        .seg_n(seg_n),
        .an_n(an_n),
        .u_tot(u_tot),
        .u_cur(u_cur),
        .u_wat(u_wat),
        .fl_disp(fl_disp),
        .clk_fl(clk_fl),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（2）电源模块 power（Verilog）

程序 2-2：power.v（部分）

```
module power(rst_n, tr_pwr, led_pwr, a_pwr, main_done, lock, clk);
    parameter TIM_CMAX = `c_ms(2000);
    parameter DEB_CMAX = `c_ms(5);
    output reg rst_n = 'b0;
```

```
    output tr_pwr;
    output led_pwr;
    input a_pwr;
    input main_done;
    input lock, clk;

    // power button with lock
    reg r_lock = 'b0;
    wire tm_done;
    wire tm_clr = !pwr; //  计时器在电源按钮未按下时无效
    // forcing poweroff
    reg r_rst_n = 'b0;
    wire pwr, pe_pwr, ne_pwr;
    // reg rst_n
    //  电源按钮触发脉冲：电源按钮下降沿、松开前后儿童锁均无效、松开前后电源状态相同
    assign tr_pwr = ne_pwr && !r_lock && !lock && r_rst_n == rst_n;
    assign led_pwr = rst_n;
    always @(posedge clk)
        if (tm_done || main_done)
            rst_n <= 'b0; //  主状态机完成或按下计时到 2 秒时电源置 0
        else if (!rst_n && tr_pwr)
            rst_n <= 'b1;
        else if (pe_pwr)
            r_rst_n <= rst_n;
    always @(posedge clk, negedge rst_n) //  沿用 button 模块的逻辑
        if (!rst_n)
            r_lock <= 'b0;
        else if (pe_pwr)
            r_lock <= lock;
    // 2 秒定时器
    timer #(
        .CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
    //  电源按钮异步输入同步化、防抖动处理
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_pwr(
        .sig(pwr),
        .pe_sig(pe_pwr),
        .ne_sig(ne_pwr),
        .a_sig(a_pwr),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

（3）儿童锁模块 protector（Verilog）

程序 2-3：protector.v（部分）

```
module protector(lock, tr_lck, led_lck, a_lck, clk, rst_n);
    parameter TIM_CMAX = `c_ms(1000);
    parameter DEB_CMAX = `c_ms(5);
    output reg lock = 'b0;
    output tr_lck;
    output led_lck;
    input a_lck;
    input clk, rst_n;

    reg r_lock = 'b0;
    wire tm_done;
    wire lck, pe_lck;
    wire tm_clr = !lck || r_lock != lock; // 在锁信号为 0 或已经改变儿童锁状态时计时器无效
    // reg lock
    assign tr_lck = tm_done;
    assign led_lck = lock;
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            lock <= 'b0;
            r_lock <= 'b0;
        end
        else if (tm_done)
            lock <= !lock; // 锁信号持续 1 秒后改变儿童锁状态
        else if (pe_lck)
            r_lock <= lock;
    // 1 秒定时器
    timer #(
        .CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
    // 锁信号异步输入同步化、防抖动处理
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_lck(
        .sig(lck),
        .pe_sig(pe_lck),
        .ne_sig(),
        .a_sig(a_lck),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（4）按钮模块 button（Verilog）

程序 2-4：button.v（部分）

```
module button(tr_btn, a_btn, lock, clk, rst_n);
    parameter DEB_CMAX = `c_ms(5);
    output tr_btn;
    input a_btn;
    input lock, clk, rst_n;

    reg r_lock = 'b0;
    wire pe_btn, ne_btn;
    assign tr_btn = ne_btn && !r_lock && !lock; // 按钮触发脉冲：按钮输入下降沿、儿童锁无效
    always @(posedge clk)
        if (!rst_n)
            r_lock <= 'b0;
        else if (pe_btn)
            r_lock <= lock; // 记录儿童锁在按下前的状态
    // 按钮异步输入同步化、防抖动处理
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_btn(
        .sig(),
        .pe_sig(pe_btn),
        .ne_sig(ne_btn),
        .a_sig(a_btn),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（5）防抖动模块 debouncer（Verilog）

程序 2-5：debouncer.v（部分）

```
module debouncer(sig, pe_sig, ne_sig, a_sig, clk, rst_n);
    parameter DEB_CMAX = `c_ms(5);
    output reg sig = 'b0; // 同步输出信号
    output pe_sig, ne_sig; // 同步信号输出的上升沿脉冲和下降沿脉冲
    input a_sig; // 异步输入信号
    input clk, rst_n;

    reg r_a_sig = 'b0;
    wire tm_done;
    wire tm_clr = sig == r_a_sig;
    // reg sig
    assign pe_sig = tm_done && !sig;
    assign ne_sig = tm_done && sig;
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            sig <= 'b0;
            r_a_sig <= 'b0;
        end
        else begin
            r_a_sig <= a_sig;
```

```
                if (tm_done) //  在状态不相等持续 5 毫秒后改变输出信号
                    sig <= !sig;
            end
        // 5 毫秒定时器
        timer #(
            .CMAX(DEB_CMAX)
        ) x_timer(
            .done(tm_done),
            .clr(tm_clr),
            .clk(clk),
            .rst_n(rst_n)
        );
endmodule
```

（6）蜂鸣器模块 buzzer 及其工具模块（Verilog）

程序 2-6：buzzer.v（部分）

```
module buzzer(buz, tr_a, tr_b, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_a, tr_b; //  触发脉冲，程序完成脉冲
    input clk, rst_n;

    wire buz_a, buz_b;
    assign buz = buz_a || buz_b; //  只要有一个需要蜂鸣器就会发响
    _buz_a #(
        .FLA_CMAX(FLA_CMAX)
    ) x_buz_a(
        .buz(buz_a),
        .tr_sig(tr_a),
        .clk(clk),
        .rst_n(rst_n)
    );
    _buz_b #(
        .FLA_CMAX(FLA_CMAX)
    ) x_buz_b(
        .buz(buz_b),
        .tr_sig(tr_b),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule

module _buz_a(buz, tr_sig, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_sig;
    input clk, rst_n;

    //    0   1   0
    //    >   +   -
    reg st = 'd0;
```

```
    wire tm_done;
    wire tm_clr = tr_sig;
    assign buz = st;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            st <= 'd0;
        else if (tr_sig)
            st <= 'd1;
        else if (tm_done && st)
            st <= 'd0;
    timer #(
        .CMAX(FLA_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule

module _buz_b(buz, tr_sig, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_sig;
    input clk, rst_n;

    //    0   1   2   3   4   5   6   7   8   9 10 11 12 13 14 15 16 17 18 19 20 21   0
    //    >   +   -   +   -   +   -   -   -   +   -   +   -   +   -   -   -   +   -   +   -   +   -
    reg [4:0] st = 'd0;
    wire tm_done;
    wire tm_clr = tr_sig;
    assign buz = st[0] && st != 7 && st != 15;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            st <= 'd0;
        else if (tr_sig)
            st <= 'd1;
        else if (tm_done && st)
            st <= st == 'd21 ? 'd0 : st + 'd1;
    timer #(
        .CMAX(FLA_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（7）LED 模块 leds 及其工具模块（Verilog）

程序 2-7：leds.v（部分）

```
module leds(
    led_dry, led_rin, led_was,
    led_fil, led_spi, led_dra,
    ld_drw, fl_drw, ld_fsd,
    clk_fl, rst_n
);
    output led_dry, led_rin, led_was;
    output led_fil, led_spi, led_dra;
    input [2:0] ld_drw, fl_drw, ld_fsd;
    input clk_fl, rst_n;

    // fwd led_dry
    // fwd led_rin
    // fwd led_was
    // fwd led_fil
    // fwd led_fil
    // fwd led_spi
    // fwd led_dra
    //  每个 LED 单独用一个子模块控制
    _led x_led_dry(
        .led(led_dry),
        .ld(ld_drw[2]),
        .fl(fl_drw[2]),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_rin(
        .led(led_rin),
        .ld(ld_drw[1]),
        .fl(fl_drw[1]),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_was(
        .led(led_was),
        .ld(ld_drw[0]),
        .fl(fl_drw[0]),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_fil(
        .led(led_fil),
        .ld(ld_fsd[2]),
        .fl('b0),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_spi(
        .led(led_spi),
        .ld(ld_fsd[1]),
        .fl('b0),
        .clk_fl(clk_fl),
```

```
        .rst_n(rst_n)
    );
    _led x_led_dra(
        .led(led_dra),
        .ld(ld_fsd[0]),
        .fl('b0),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
endmodule

module _led(led, ld, fl, clk_fl, rst_n);
    output led;
    input ld, fl;
    input clk_fl, rst_n;

    // 电源关闭时不输出，要求闪烁时使用 1 Hz 时钟信号
    assign led = rst_n && (fl ? clk_fl : ld);
endmodule
```

（8）数码管模块 display 及其工具模块（Verilog）

程序 2-8：display.v（部分）

```
module display(seg_n, an_n, u_tot, u_cur, u_wat, fl_disp, clk_fl, clk, rst_n);
    parameter SCA_CMAX = `c_ms(1);
    output [7:0] seg_n, an_n;
    input [5:0] u_tot, u_cur, u_wat;
    input fl_disp;
    input clk_fl, clk, rst_n;

    wire clk_sca; // 扫描用时钟信号
    wire [3:0] mem[7:0]; // 8 个存储单元，每个 4 位，用来存储待显示的数字（十进制）
    // 第 2、3 位常灭
    assign mem[3] = 'b1111;
    assign mem[2] = 'b1111;
    wire [2:0] pos;
    // fwd seg_n
    assign an_n = ~((rst_n && (!fl_disp || clk_fl)) << pos);
    // 提取需要显示数字的十位和个位
    _disp_decimal x_dec_tot(
        .e1(mem[7]),
        .e0(mem[6]),
        .val(u_tot)
    );
    _disp_decimal x_dec_cur(
        .e1(mem[5]),
        .e0(mem[4]),
        .val(u_cur)
    );
    _disp_decimal x_dec_wat(
        .e1(mem[1]),
```

```
        .e0(mem[0]),
        .val(u_wat)
    );
    // 周期 5 ms 分频器
    divider #(
        .CMAX(SCA_CMAX)
    ) x_divider(
        .clk_div(clk_sca),
        .clk(clk),
        .rst_n(rst_n)
    );
    // 模 8 计数器
    _disp_counter8 x_counter(
        .cnt(pos),
        .clk(clk_sca),
        .rst_n(rst_n)
    );
    // 七段式数码管译码器
    _disp_pattern x_pattern(
        .seg_n(seg_n),
        .val(mem[pos])
    );
endmodule

module _disp_decimal(e1, e0, val);
    output [3:0] e1, e0;
    input [5:0] val;
    assign e1 = val / 10;
    assign e0 = val % 10;
endmodule

module _disp_counter8(cnt, clk, rst_n);
    output reg [2:0] cnt = 'd0;
    input clk, rst_n;

    // reg cnt
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            cnt <= 'd0;
        else
            cnt <= cnt + 'd1;
endmodule

module _disp_pattern(seg_n, val);
    output reg [7:0] seg_n; // combinational
    input [3:0] val;
    always @(*)
        case (val)
            'd0:    seg_n <= 'b11000000;
            'd1:    seg_n <= 'b11111001;
            'd2:    seg_n <= 'b10100100;
```

```
            'd3:      seg_n <= 'b10110000;
            'd4:      seg_n <= 'b10011001;
            'd5:      seg_n <= 'b10010010;
            'd6:      seg_n <= 'b10000010;
            'd7:      seg_n <= 'b11111000;
            'd8:      seg_n <= 'b10000000;
            'd9:      seg_n <= 'b10010000;
            default: seg_n <= 'b11111111;
        endcase
endmodule
```

（9）主状态机模块 main（Verilog）

程序 2-9：main.v（部分）

```
module main(
    done, prog_done,
    ioh,
    ld_drw, fl_drw, ld_fsd,
    u_tot, u_cur, u_wat, fl_disp,
    tr_pwr, tr_mod, tr_run, tr_wat,
    clk, rst_n
);
    parameter TIM_CMAX = `c_ms(1000);
    parameter END_CMAX = `c_s(10);
    output done, prog_done;
    output ioh;
    output [2:0] ld_drw, fl_drw;
    output [2:0] ld_fsd;
    output [5:0] u_tot, u_cur, u_wat;
    output fl_disp;
    input tr_pwr, tr_mod, tr_run, tr_wat;
    input clk, rst_n;

    // 进入设置的触发脉冲：模式选择或水位选择任一
    wire tr_set = tr_mod || tr_wat;

    // 状态：'b00 设置，'b01 运行，'b10 待关机，'b11 暂停
    reg [1:0] st = 'b00;
    wire started = st[0]; // 最低位为 1 时就是正在洗衣
    wire se_done;
    wire [2:0] se_ld_drw;
    wire [5:0] se_u_tot, se_u_cur, se_u_wat;
    wire [2:0] se_mode;
    wire se_clr = started;
    wire [2:0] ru_ld_drw, ru_fl_drw;
    wire [2:0] ru_ld_fsd;
    wire [5:0] ru_u_tot, ru_u_cur;
    wire ru_done;
    wire ru_pau = st == 'b11;
    wire ru_clr = !started;
    wire tm_done;
    wire tm_clr = st != 'b10;
```

```
// 主状态机完成脉冲：等待关机状态下定时器完成或没洗衣的时候接到电源触发脉冲
assign done = rst_n && ((st == 'b10 && tm_done) || (!started && tr_pwr));
// fwd prog_done
assign ioh = started;
assign ld_drw = started ? ru_ld_drw : se_ld_drw;
assign fl_drw = started ? ru_fl_drw : 'b000;
assign ld_fsd = started ? ru_ld_fsd : 'b000;
assign u_tot = started ? ru_u_tot : se_u_tot;
assign u_cur = started ? ru_u_cur : se_u_cur;
assign u_wat = se_u_wat;
assign fl_disp = ru_pau;
always @(posedge clk, negedge rst_n)
    if (!rst_n)
        st <= 'b00;
    else case (st)
        'b00: // setting
            if (se_done)
                st <= 'b01;
        'b01: // running
            if (ru_done)
                st <= 'b10;
            else if (tr_run)
                st <= 'b11;
        'b10: // waiting_end
            if (tr_set)
                st <= 'b00;
            else if (se_done)
                st <= 'b01;
        'b11: // paused
            if (tr_run)
                st <= 'b01;
            else if (tr_set)
                st <= 'b00;
    endcase
setting x_setting(
    .done(se_done),
    .mode(se_mode),
    .ld_drw(se_ld_drw),
    .u_tot(se_u_tot),
    .u_cur(se_u_cur),
    .u_wat(se_u_wat),
    .tr_mod(tr_mod),
    .tr_run(tr_run),
    .tr_wat(tr_wat),
    .clr(se_clr),
    .clk(clk),
    .rst_n(rst_n)
);
run_mode #(
    .TIM_CMAX(TIM_CMAX)
) x_run_mode(
    .done(ru_done),
```

```
        .prog_done(prog_done),
        .ld_drw(ru_ld_drw),
        .fl_drw(ru_fl_drw),
        .ld_fsd(ru_ld_fsd),
        .u_tot(ru_u_tot),
        .u_cur(ru_u_cur),
        .u_wat(se_u_wat),
        .init(se_mode),
        .pau(ru_pau),
        .clr(ru_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
    timer #(
        .CMAX(END_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（10）设置模块 setting（Verilog）

程序 2-10：setting.v（部分）

```
module setting(
    done, mode,
    ld_drw,
    u_tot, u_cur, u_wat,
    tr_mod, tr_run, tr_wat,
    clr, clk, rst_n
);
    output done;
    output reg [2:0] mode = 'b111;
    output [2:0] ld_drw;
    output [5:0] u_tot, u_cur;
    output reg [5:0] u_wat = 'd3;
    input tr_mod, tr_run, tr_wat;
    input clr, clk, rst_n;

    // drw =>     w =>    rw =>    r    => dr    => d      => drw
    // 111       001       011       010       110       100       111
    assign done = tr_run; //  设置完成：设置状态下接到启动/暂停触发脉冲
    // reg mode
    assign ld_drw = mode;
    assign u_tot = u_wat * (
        (mode[0] ? `TG_WAS : 0) +
        (mode[1] ? `TG_RIN : 0) +
        (mode[2] ? `TG_DRY : 0)
    ); //  计算总时间，TG_*为系数，乘上水位即为模式总时间
    assign u_cur = u_wat * (
```

```
            mode[0] ? `TG_WAS :
            mode[1] ? `TG_RIN :
            mode[2] ? `TG_DRY : 0
        ); // 计算第一个程序的时间
        // reg u_wat
        always @(posedge clk, negedge rst_n)
            if (!rst_n) begin
                mode <= 'b111;
                u_wat <= 'd3;
            end
            else if (clr)
                mode <= 'b111;
            else if (tr_mod)
                case (mode) // 使用 3 位二进制掩码表示模式，依次切换
                    'b111:   mode <= 'b001;
                    'b001:   mode <= 'b011;
                    'b011:   mode <= 'b010;
                    'b010:   mode <= 'b110;
                    'b110:   mode <= 'b100;
                    'b100:   mode <= 'b111;
                    default: mode <= 'b111;
                endcase
            else if (tr_wat)
                u_wat <= u_wat == 'd5 ? 'd2 : u_wat + 'd1;
endmodule
```

（11）模式运行模块 run_mode（Verilog）

程序 2-11：run_mode.v（部分）

```
module run_mode(
        done, prog_done,
        ld_drw, fl_drw, ld_fsd,
        u_tot, u_cur,
        u_wat, init, pau, clr, clk, rst_n
);
        parameter TIM_CMAX = `c_ms(1000);
        output done, prog_done;
        output [2:0] ld_drw, fl_drw;
        output [2:0] ld_fsd;
        output [5:0] u_tot, u_cur;
        input [5:0] u_wat;
        input [2:0] init; // 所设置的模式
        input pau, clr, clk, rst_n;

        reg [2:0] st = 'b000;
        wire [2:0] lowb = st & -st; // 获取最低位，即当前程序
        wire [2:0] srem = st ^ lowb; // 剩余程序
        reg r_prog_done = 'b0;
        reg [4:0] prog_init; // combinational
        always @(*)
            case (lowb) // 根据当前程序设置所需的过程，使用 5 位二进制掩码
```

```
                'b001:      prog_init <= 'b01100;
                'b010:      prog_init <= 'b10111;
                'b100:      prog_init <= 'b00011;
                default: prog_init <= 'b00000;
            endcase
        reg r_clr = 'b0;
        wire prog_clr = r_clr || r_prog_done;
        assign done = !st && r_prog_done; //  模式完成脉冲：没有需要执行的程序且接到程序完成脉冲
        // fwd prog_done
        assign ld_drw = st;
        assign fl_drw = lowb;
        // fwd ld_fsd
        assign u_tot = u_cur + u_wat * (
            (srem[0] ? `TG_WAS : 0) +
            (srem[1] ? `TG_RIN : 0) +
            (srem[2] ? `TG_DRY : 0)
        ); //  计算总剩余时间
        // fwd u_cur //  当前程序剩余时间由程序运行模块决定
        always @(posedge clk, negedge rst_n)
            if (!rst_n) begin
                st <= 'b000;
                r_prog_done <= 'b0;
                r_clr <= 'b0;
            end
            else if (!pau) begin //  暂停状态下不更新状态
                r_prog_done <= prog_done;
                r_clr <= clr;
                if (clr) begin
                    st <= init;
                end
                else if (prog_done)
                    st <= srem;
            end
        run_prog #(
            .TIM_CMAX(TIM_CMAX)
        ) x_run_prog(
            .done(prog_done),
            .ld_fsd(ld_fsd),
            .u_cur(u_cur),
            .u_wat(u_wat),
            .init(prog_init),
            .pau(pau),
            .clr(prog_clr),
            .clk(clk),
            .rst_n(rst_n)
        );
    endmodule
```

（12）程序运行模块 run_prog（Verilog）

程序 2-12：run_prog.v（部分）

```
module run_prog(
```

```
    done,
    ld_fsd, u_cur,
    u_wat, init, pau, clr, clk, rst_n
);
    parameter TIM_CMAX = `c_ms(1000);
    output done;
    output [2:0] ld_fsd;
    output [5:0] u_cur;
    input [5:0] u_wat;
    input [4:0] init;
    input pau, clr, clk, rst_n;

    reg[4:0] st = 'b00000; // 使用 5 位二进制掩码表示所需过程
    wire [4:0] lowb = st & -st; // 取最低位为当前执行的过程
    wire [4:0] srem = st ^ lowb; // 剩余过程
    reg [5:0] tm_init_mul; // combinational
    always @(*)
        case (lowb) // 根据最低位决定时间系数
            'b00001: tm_init_mul <= `TC_DRA;
            'b00010: tm_init_mul <= `TC_SPI;
            'b00100: tm_init_mul <= `TC_FIL;
            'b01000: tm_init_mul <= `TC_WAS;
            'b10000: tm_init_mul <= `TC_RIN;
            default: tm_init_mul <= 'd0;
        endcase
    wire tm_done;
    reg r_tm_done = 'b0;
    wire [5:0] tm_rema;
    wire [5:0] tm_init = u_wat * tm_init_mul;
    reg r_clr = 'b0;
    wire tm_clr = r_clr || r_tm_done;
    assign done = !st && r_tm_done; // 程序完成脉冲：没有要执行的过程且接到定时器完成脉冲
    assign ld_fsd = lowb[2:0];
    assign u_cur = tm_rema + u_wat * ( // 计算当前程序剩余时间
        (srem[0] ? `TC_DRA : 0) +
        (srem[1] ? `TC_SPI : 0) +
        (srem[2] ? `TC_FIL : 0) +
        (srem[3] ? `TC_WAS : 0) +
        (srem[4] ? `TC_RIN : 0)
    );
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            st <= 'b00000;
            r_tm_done <= 'b0;
            r_clr <= 'b0;
        end
        else if (!pau) begin // 暂停时不更新状态
            r_tm_done <= tm_done;
            r_clr <= clr;
            if (clr)
                st <= init;
```

```
            else if (tm_done)
                st <= srem;
        end
    timer_nested #(
        .CBIT(6),
        .INNER_CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .rema(tm_rema),
        .init(tm_init),
        .pau(pau),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

（13）嵌套定时器模块 timer_nested（Verilog）

程序 2-13：timer_nested.v（部分）

```
module timer_nested(done, rema, init, pau, clr, clk, rst_n);
    parameter CBIT = 6;
    parameter INNER_CMAX = `c_ms(1000);
    output done;
    output reg [CBIT - 1:0] rema = 'd0; // 倒计时（单位：秒）
    input [CBIT - 1:0] init; // 定时（单位：秒）
    input pau, clr, clk, rst_n;

    wire ok = !rema; // 剩余 0 秒时倒计时结束
    wire tm_done;
    reg r_tm_done = 'b0;
    assign done = r_tm_done && ok; // 完成脉冲：倒计时结束与延后一周期的定时器脉冲取与
    // reg rema
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            rema <= 'd0;
            r_tm_done <= 'b0;
        end
        else if (!pau) begin // 暂停不更新状态
            r_tm_done <= tm_done;
            if (clr)
                rema <= init;
            else if (tm_done)
                rema <= (ok ? init : rema) - 1; // 倒数计时
        end
    timer #(
        .CMAX(INNER_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(clr),
        .clk(clk),
```

```
        .rst_n(rst_n)
    );
endmodule
```

（14）定时器模块 timer（Verilog）

程序 2-14：timer.v（部分）

```
module timer(done, clr, clk, rst_n);
`include "h_cbit.v" // 提供 cbit 函数
    parameter CMAX = `c_ms(1000);
    localparam CBIT = cbit(CMAX); //在 Static Elaborate 阶段得到计数所需要的二进制位数
    output done;
    input clr, clk, rst_n;

    reg [CBIT - 1:0] cnt = 'd0;
    assign done = cnt == CMAX; // 完成脉冲：在计数达到指定值时输出脉冲
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            cnt <= 'd0;
        else if (clr)
            cnt <= 'd0;
        else
            cnt <= done ? 'd1 : cnt + 'd1;
endmodule
```

（15）分频器模块 divider（Verilog）

程序 2-15：divider.v（部分）

```
module divider(clk_div, clk, rst_n);
`include "h_cbit.v"
    parameter CMAX = `c_ms(500); // 半周期
    localparam CBIT = cbit(CMAX);
    output reg clk_div = 'b1; // 分频后时钟信号初始为 1
    input clk, rst_n;

    reg [CBIT - 1:0] cnt = 'd0;
    // reg clk_div
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            clk_div <= 'b1;
            cnt <= 'd0;
        end
        else if (cnt == CMAX) begin
            clk_div <= !clk_div; // 计数到指定值时将分频时钟信号取反
            cnt <= 'd1;
        end
        else
            cnt <= cnt + 'd1;
endmodule
```

2.5 仿真过程

为了验证设计的正确性，对模块 debouncer、buzzer、setting、run_mode 和顶层模块 top 进行了仿真。为了方便仿真代码编写，使用一个仿真公用头文件，提供时间-计数转换宏、延时转换宏和系统时钟信号的定义，具体代码见附录。

（1）顶层模块仿真

目的：验证系统除数码管模块、防抖动模块和蜂鸣器模块的验证外的功能。

输入：a_mod（模式选择），a_run（启动/暂停），a_wat（水位选择），a_pwr（电源）。

输出：ioh（舱门锁），led_pwr（电源），led_lck（儿童锁），led_dry（脱水模式），led_rin（漂洗模式），led_was（洗涤模式），led_fil（进水过程），led_spi（甩干过程），led_dra（排水过程）。

变量：u_tot（总剩余时间），u_cur（当前程序剩余时间），u_wat（水位）。

顶层模块仿真代码（Verilog）如下：

程序 2-16：test/tb_top.v

```
`timescale 1ns / 1ps

module tb_top();
`include "tb_h_common.v"
`define press(x) begin `us(2) x = 1; `us(1) x = 0; end
    wire pwr;
    reg a_mod = 0;
    reg a_run = 0;
    reg a_wat = 0;
    reg a_pwr = 0;
    initial begin
        `press(a_pwr); // 开机
        repeat (6) `press(a_mod); // 依次切换模式
        repeat (4) `press(a_wat); // 依次切换水位
        `press(a_run); // 以“洗漂脱”模式和 3 kg 衣量对应水位开始运行
        `us(20) a_mod = 1; // 按住模式选择
        `us(4)  a_wat = 1; // 按住水位选择
        `us(20) a_wat = 0; // 松开水位选择，松开前儿童锁开启
        `us(6)  a_mod = 0; // 松开模式选择
        `us(20) a_pwr = 1; // 按下电源
        `us(30) a_pwr = 0; // 松开电源，松开前强制断电
        `us(50) `press(a_pwr); // 开机
        `press(a_run); // 运行
        `us(40) `press(a_run); // 暂停
        `us(10) `press(a_pwr); // 不响应
        `us(10) `press(a_run); // 继续运行
```

```
        `us(20) a_mod = 1; // 按住模式选择
        `us(4)  a_wat = 1; // 按住水位选择
        `us(12) a_mod = 0; // 松开模式选择，松开前儿童锁开启
        `us(2)  a_wat = 0; // 松开水位选择
        `press(a_pwr); // 不响应
        `press(a_mod); // 不响应
        `press(a_run); // 不响应
        `press(a_wat); // 不响应
        `us(20) a_mod = 1; // 按下模式选择
        `us(4)  a_wat = 1; // 按下水位选择
        `us(12) a_mod = 0; // 松开模式选择，松开前儿童锁关闭
        `us(4)  a_wat = 0; // 松开水位选择
        `press(a_run); // 暂停
        `press(a_pwr); // 不响应
        repeat (5) `press(a_mod); // 重新设置模式
        `press(a_run); // 运行，直到自动关机
    end
    top #(
        .TIM_UNIT(`c_us(10)),
        .END_WAIT(`c_us(100)),
        .DEB_WAIT(`c_cp(5)),
        .LCK_WAIT(`c_us(10)),
        .FPO_WAIT(`c_us(20)),
        .BUZ_INTV(`c_us(1)),
        .FLA_INTV(`c_us(5)),
        .DIS_INTV(`c_cp(1))
    ) x_dut(
        .led_pwr(pwr),
        .a_mod(a_mod),
        .a_run(a_run),
        .a_wat(a_wat),
        .a_pwr(a_pwr),
        .clk(clk)
    );
endmodule
```

为了方便在短时间内完成仿真任务，将倒计时的时间单位由 1 s 改为 10 us，同时洗衣时间等比例缩小，关机前等待时间改为 100 us，同时按下按钮后儿童锁触发前的等待时间改为 10 us，电源按钮按下后强制断电前的等待时间改为 20 us。程序指示灯的闪烁周期由 1 s 改为 10 us。

第一部分仿真图如图 2-4 所示。

a. 电源最初关闭，按下电源按钮，电源打开，进入设置，默认模式为“洗漂脱”（三个程序指示灯全亮），默认水位对应 3 kg 衣量。模式总时间为 33，第一个程序时间为 12，水位显示 3。

b. 按六次模式选择按钮，模式由“洗漂脱”依次变为“单洗”、“洗漂”、“单漂”、“漂脱”、“单脱”，最后变回“洗漂脱”。期间，水位不变，模式总时间和第一个程序时间以及各

程序指示灯变化符合规则。

c. 当前仍为“洗漂脱”模式，按四次水位选择按钮，水位对应的衣量由 3 kg 依次变为 4 kg、5 kg、2 kg，最后变回 3 kg。期间，水位变化符合上述循环，模式总时间和第一个程序时间始终与水位成正比，符合规则。

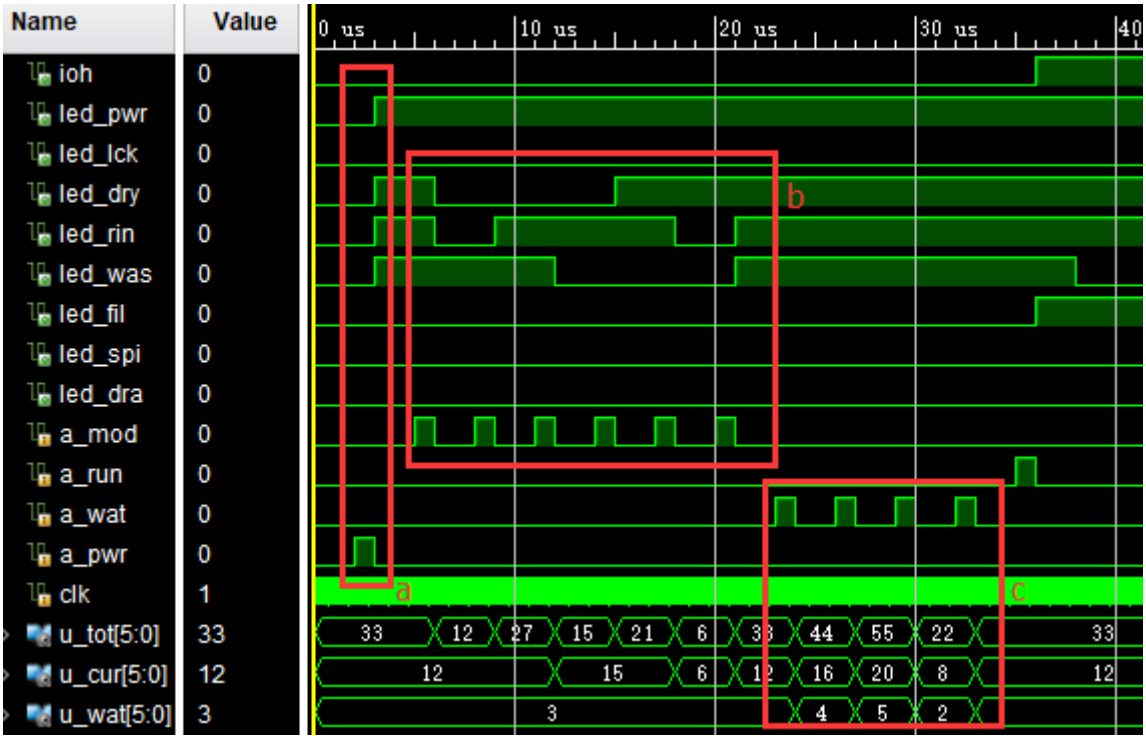

图 2-4 顶层模块仿真图（一）

第二部分仿真图如图 2-5 所示。

d. 设置完毕后，按下启动/暂停按钮，洗衣机开始洗衣，舱门上锁。

e. 同时长按模式选择按钮和水位选择按钮的 10 us 后儿童锁启动，儿童锁指示灯亮。

f. 在儿童锁开启状态下，长按电源按钮两秒后电源关闭，系统不输出。

第三部分仿真图如图 2-6 所示。

g. 重新按下电源按钮开机，按下启动/暂停按钮，洗衣机开始洗衣，舱门锁启动。

h. 倒计时开始正常工作，仿真时以 10 us 为单位。

i. 当前程序（洗涤）指示灯闪烁，未完成程序指示灯继续常亮，前 30 us 进水过程时进水过程指示灯亮。

j. 按下启动/暂停按钮暂停洗衣状态，倒计时停止。运行和暂停时均不响应电源按钮的触发脉冲。再次按下暂停按钮，洗衣过程继续。

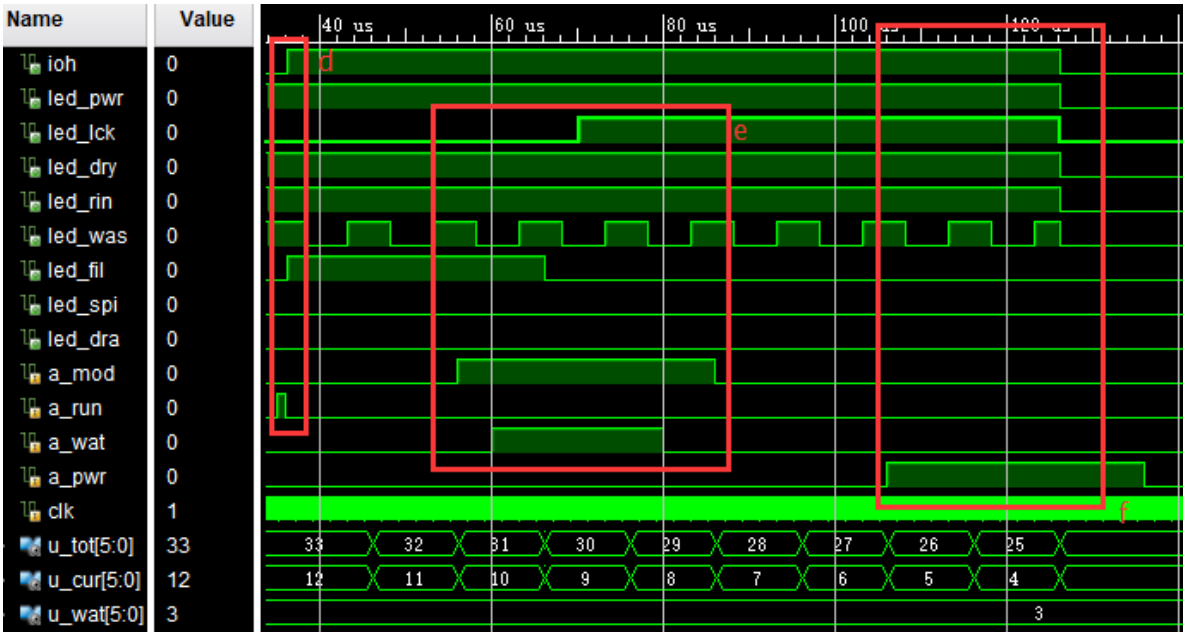

图 2-5 顶层模块仿真图（二）

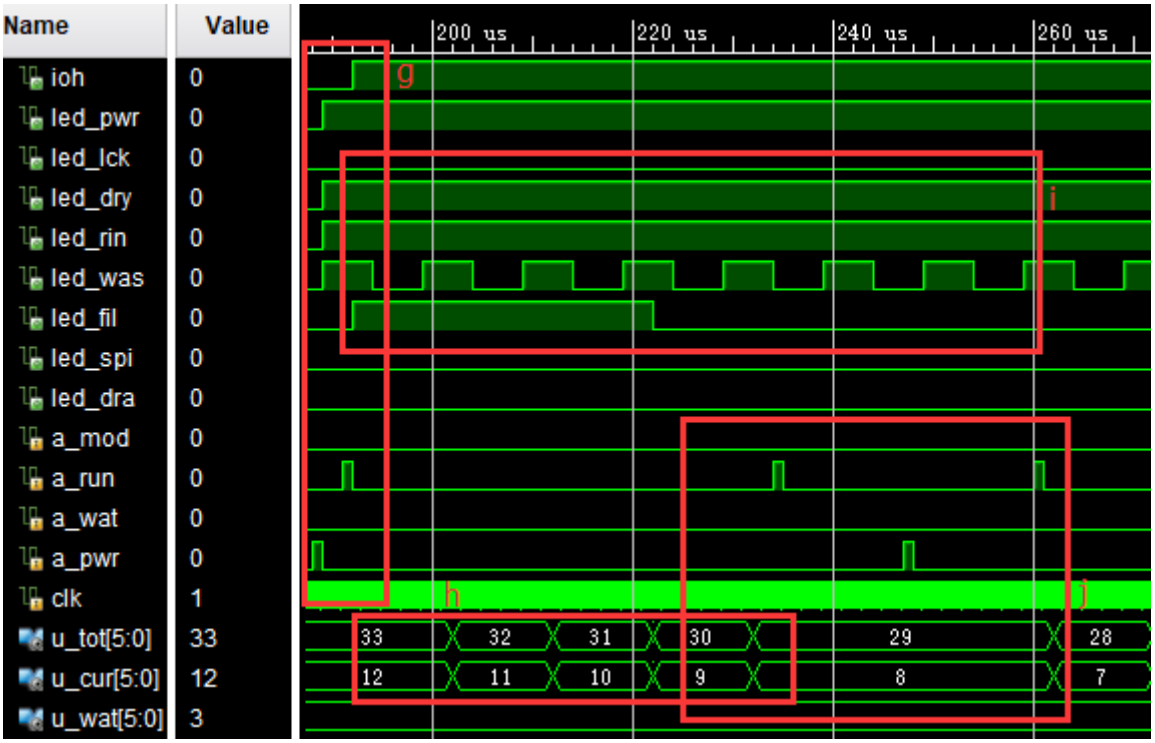

图 2-6 顶层模块仿真图（三）

第四部分仿真图如图 2-7 所示。

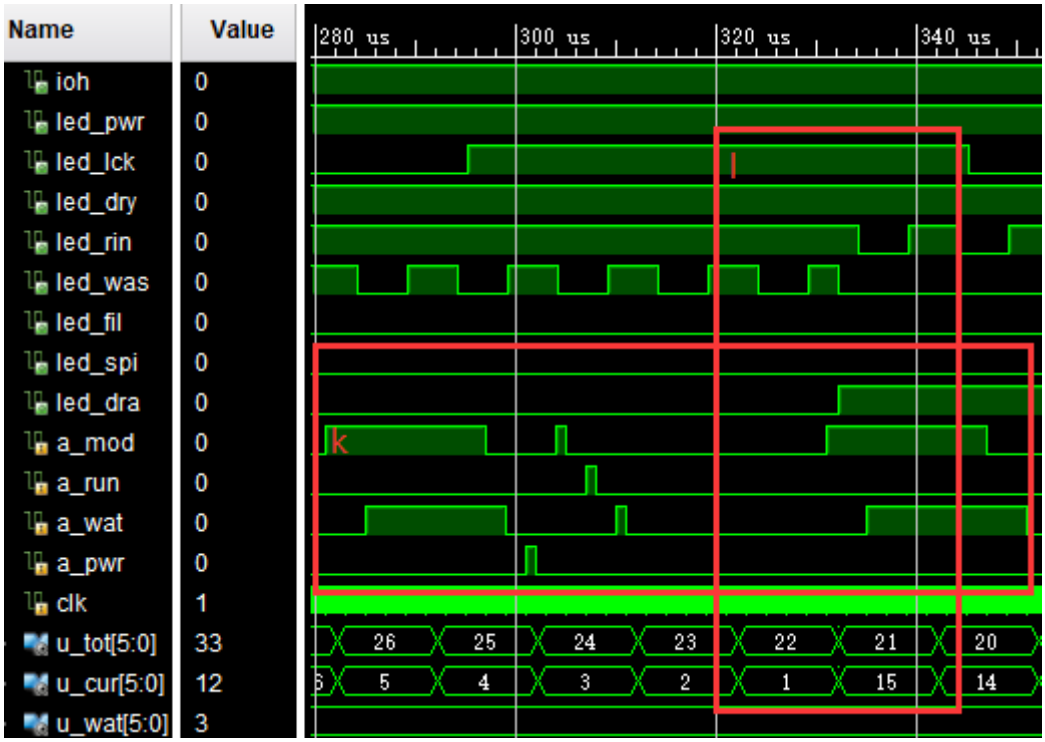

图 2-7 顶层模块仿真图（四）

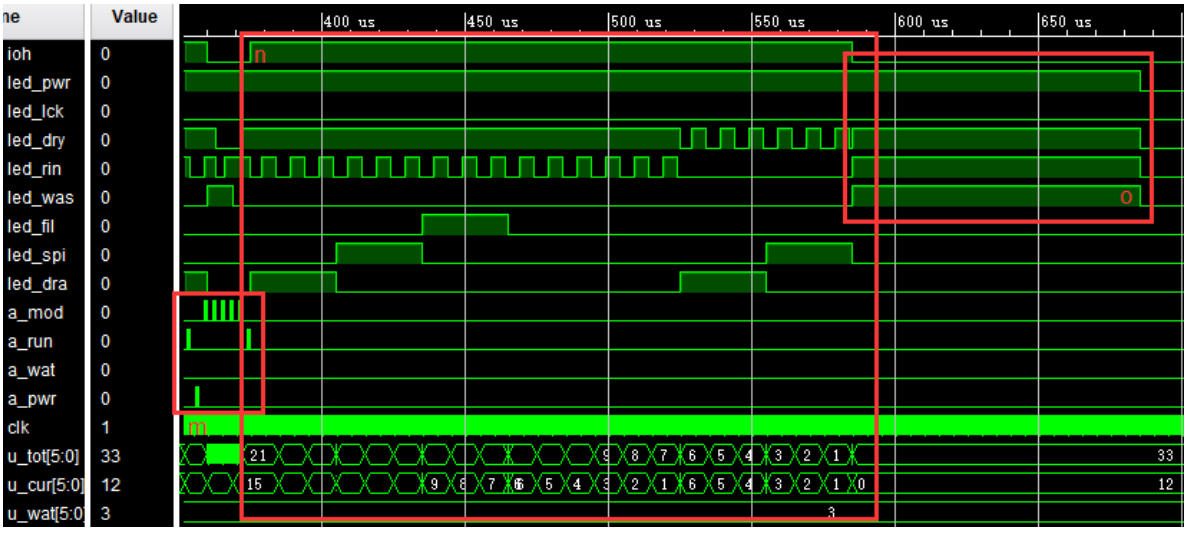

图 2-8 顶层模块仿真图（五）

k. 同时按下模式选择按钮、水位选择按钮的 10 us 后，儿童锁启动，四个按钮的触发脉冲均不响应。再次同时按下模式选择按钮、水位选择按钮，10 us 后儿童锁关闭。

l. 这时洗涤程序结束，进入漂洗程序，洗涤指示灯灭，漂洗指示灯开始闪烁，漂洗程序的第一个过程是排水，排水指示灯亮，模式总剩余时间和当前程序剩余时间被重新计算。

第五部分仿真图如图 2-7 所示。

m. 暂停洗衣，不响应电源按钮触发脉冲，重新选择模式为“漂脱”，再按启动/暂停开始新的洗衣。

n. 水位对应 3 kg 衣量时，“漂脱”模式一共 21 个单位，其中漂洗过程 15 个单位，脱水 6 个时间单位。210 us 后，洗衣结束。

o. 洗衣结束后等待 100 us，电源自动关闭。

（2）防抖动模块仿真

目的：验证防抖动模块的功能，包括输出稳定同步信号、输出上升沿脉冲、输出下降沿脉冲。

输入：a_sig（异步信号），clk（系统时钟），rst_n（复位信号）。

输出：sig（同步信号），pe_sig（上升沿脉冲），ne_sig（下降沿脉冲）。

防抖动模块仿真代码（Verilog）如下：

程序 2-17：test/tb_debouncer.v

```
`timescale 1ns / 1ps

module tb_debouncer();
`include "tb_h_common.v"
    reg a_sig = 'b0;
    initial begin
        `cp(1.3)    a_sig = 1;
        `cp(10.1) a_sig = 0; //  响应，sig 设为 1
        `cp(9.9)    a_sig = 1; //  不响应，sig 仍为 1
        `cp(2.8)    a_sig = 0;
        `cp(10.1) a_sig = 1; //  响应，sig 设为 0
        `cp(9.9)    a_sig = 0; //  不响应，sig 仍为 0
    end
    debouncer #(
        .DEB_CMAX(`c_cp(10)) //  防抖动时间 10 周期，即 0.1 微秒
    ) x_dut(
        .a_sig(a_sig),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

防抖动阈值设为 10 个时钟周期，即为 100 ns。

仿真图如图 2-9 所示。

a. 在 a_sig 置 1 后 10.1 个时钟周期又将 a_sig 置为 0，该 1 信号应视为有效。在置 1 的 10 个周期后同步地输出了长度为一个周期的上升沿脉冲，并将同步信号输出置为 1。

b. 在 a_sig 置 0 后 9.9 个时钟周期又将其置为 1，该 0 信号应视为无效，不改变同步信号输出，也不输出下降沿脉冲。

c. 在 a_sig 置 0 后 10.1 个周期又将其置为 1，该 0 信号应视为有效。在置 0 的 10 个周期后同步地输出了一个长度为一个周期的下降沿脉冲，并将同步信号输出置为 0。

d. 在 a_sig 置 1 后 9.9 个周期又将其置为 0，该 1 信号应视为无效，不改变同步信号输出，也不输出上升沿脉冲。

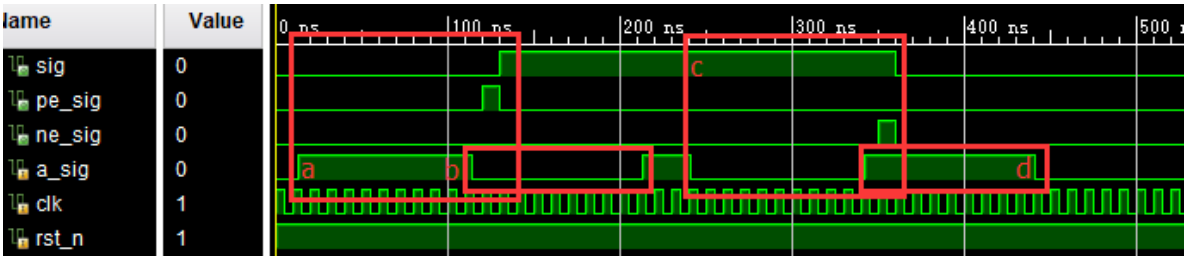

图 2-9 防抖动模块仿真图

（3）蜂鸣器模块仿真

目的：验证蜂鸣器的功能，包括对触发脉冲的响应和程序完成脉冲的响应。

输入：tr_a（触发脉冲），tr_b（程序完成脉冲），clk（系统时钟），（复位信号）。

输出：buz（蜂鸣器）。

防抖动模块仿真代码（Verilog）如下：

程序 2-18：test/tb_buzzer.v

```
`timescale 1ns / 1ps

module tb_buzzer();
`include "tb_h_common.v"
`define press(x) begin `us(2) x = 1; `us(1) x = 0; end
    reg a_a = 0; //  模拟按钮触发
    reg a_b = 0; //  模拟程序完成
    initial begin
        `press(a_a);
        `us(10)   `press(a_b);
        `us(4.7) `press(a_a);
        `us(120) `press(a_b);
        `us(12)   `press(a_a);
    end
    wire tr_a, tr_b;
    buzzer #(
        .FLA_CMAX(`c_us(5))
    ) x_dut(
        .tr_a(tr_a),
        .tr_b(tr_b),
        .clk(clk),
        .rst_n('b1)
    );
```

```
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_a(
        .tr_btn(tr_a),
        .a_btn(a_a),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_b(
        .tr_btn(tr_b),
        .a_btn(a_b),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

使用两个按钮模块用于产生两种脉冲，单次蜂鸣持续时间为 5 us。

仿真图如图 2-10 所示。

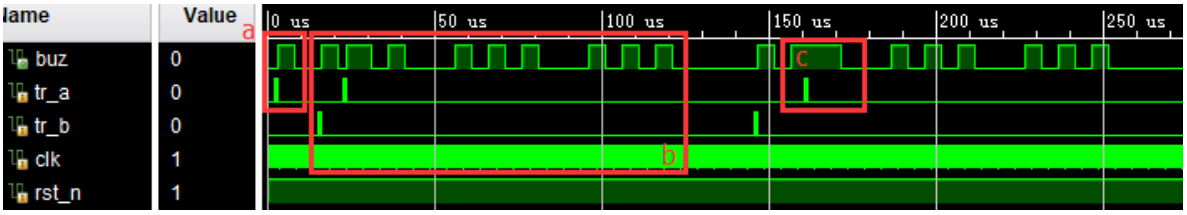

图 2-10 蜂鸣器模块仿真图

a. 接到触发脉冲，蜂鸣器“嘟”响一次。

b. 接到程序完成脉冲，蜂鸣器按照模式“嘟嘟嘟”、“嘟嘟嘟”、“嘟嘟嘟”响。

c. 在两种信号只要有一个信号需要蜂鸣器响时，蜂鸣器就会响。

2.6 主要故障

基本没有故障，但是注意到一点在实现设计时会给出警告。

（1）实现设计时会给出警告

问题描述：实现时会给出警告[DRC CFGBVS-1]。

警告信息：“[DRC CFGBVS-1] Missing CFGBVS and CONFIG_VOLTAGE Design Properties: Neither the CFGBVS nor CONFIG_VOLTAGE voltage property is set in the current_design. Configuration bank voltage select (CFGBVS) must be set to VCCO or GND, and CONFIG_VOLTAGE must be set to the correct configuration voltage, in order to determine the I/O voltage support for the pins

in bank 0.”

问题分析：经过查阅资料，得知这是设计规范检查时报的警告，该警告是要求设置 CFGBVS 和 CONFIG_VOLTAGE。CFGBVS 应设为 VCCO 或 GND。CONFIG_VOLTAGE 根据该开发板，应设为 3.3 V。

解决方法：在约束文件中添加两句“set_property CFGBVS VCCO [current_design];”和“set_property CONFIG_VOLTAGE 3.3 [current_design];”，将 CFGBVS 和 CONFIG_VOLTAGE 设为正确值。

2.7 功能测试

共进行了 3 项测试，它们分别为：设置、洗衣、舱门锁、暂停/恢复、自动关机功能测试，暂停并设置、等待自动关机时操作、强制断电功能测试，防误关机、儿童锁功能测试。测试过程中同时留意各 LED 和数码管的输出。

（1）设置、洗衣、舱门锁、暂停/恢复、自动关机功能测试

按下电源按钮开机，进入设置，默认设置如图 2-11，为“洗漂脱”模式和对应 3 kg 衣量的水位，洗涤程序、漂洗程序、脱水程序指示灯常亮，总时间显示 33 秒，第一步时间显示 12 秒，水位显示对应 3 kg 衣量。

多次多次按下模式选择按钮和水位选择按钮，如图 2-12，最终选择“漂脱”模式和对应 4 kg 衣量的水位。

图 2-11 开机后初始状态

图 2-12 “漂脱”模式和 4 kg 衣量对应的水位

按下启动/暂停按钮，开始洗衣，舱门锁开启，各指示灯行为符合规则，数码管上水位显示 4 不变，倒计时每一秒跳动一次，与现实时间相符。

十数秒后再次按下启动/暂停按钮，倒计时停止，数码管由常亮改为闪烁。数秒后再次按下启

动/暂停按钮，洗衣继续。各指示灯及数码管行为均符合规则。漂洗过程和脱水过程结束时，蜂鸣器均按照“嘟嘟嘟”、“嘟嘟嘟”、“嘟嘟嘟”发响，倒计时的当前程序剩余时间部分重置为新程序的总时间。洗衣结束后状态如图 2-13，回到设置，模式默认回到“洗漂脱”，水位不变。无操作十秒后，如图 2-14，系统自动关闭，各指示灯及数码管均熄灭。

图 2-13 洗衣结束状态

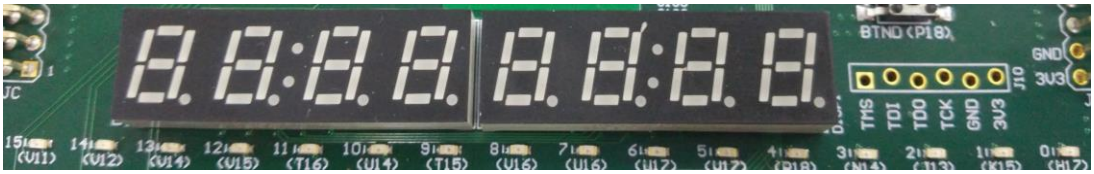

图 2-14 系统自动关闭后的状态

综上，该功能测试通过，无异常行为。

（2）暂停并设置、等待自动关机时操作、强制断电功能测试

按下电源按钮开机，再按下启动/暂停按钮开始洗衣。

数秒后按下暂停，再按下模式选择按钮，进入设置，将模式设置为“洗漂”模式再按下启动/暂停按钮，重新开始洗衣，倒计时与新模式设置相符。

洗衣结束后直接按下启动/暂停按钮，开始“洗漂脱”模式，等待十数秒，系统没有自动关闭，洗衣正常。

此时长按电源按钮，两秒后系统关闭，所有指示灯、数码管熄灭。

综上，该功能测试通过，无异常行为。

（3）防误关机、儿童锁功能测试

开机，默认“洗漂脱”模式和 3 kg 衣量对应的水位，选择时间最长的“洗漂脱”模式和 5 kg 衣量对应的水位，方便后续测试。

按下启动/暂停按钮，开始洗衣。此时随意按下电源按钮、模式选择按钮、水位选择按钮，系统仍正常洗衣，不响应按钮输入。

长按模式选择按钮和水位选择按钮，1 秒后蜂鸣器“嘟”发响，如图 2-15，儿童锁指示灯亮。此时按下启动/暂停按钮，系统不响应。当洗衣结束后，按任何按钮均不响应。

图 2-15 儿童锁开启

综上，该测试通过，无异常行为。

2.8 实验中遇到的主要问题及解决方法

表 2-3 主要问题及解决方法

问题	解决方案
如何让 LED 和数码管闪烁	用分频器获取 1 Hz 时钟信号，在相应的输出模块中在 1 Hz 时钟信号为高电平时输出有效，低电平时不输出。
如何在数码管上同时显示多个数字	根据开发板文档，由 AN 掩码控制当前有效的位，以 1 ms 为间隔，AN 不断从'b11111110 开始循环向左移位，就可以在人眼响应时间内实现同时显示。
如何使蜂鸣器按照“嘟嘟嘟”、“嘟嘟嘟”、“嘟嘟嘟”发响	使用一个 0~21 的计数器，初始为 0，接收到程序完成脉冲时变为 1，然后每 0.1 秒增加 1，21 时变为 0。用一个寄存器来保存当前蜂鸣器是否发响，初始为 0，计数器为 1、3、5、9、11、13、17、19、21 时设为 1，否则设为 0。
如何实现异步输入信号的同步化、防抖动处理	使用两个寄存器，初值均为 0，第二个寄存器作为输出。在时钟上升沿时将异步输入信号赋给第一个寄存器，第一个寄存器原值赋给第二个寄存器。如果第二个寄存器和第一个寄存器值不同，可能需要将输出信号取反，也有可能是信号的不稳定部分。此时启动一个定时器，如果在定时器结束后两个寄存器值还是不同，就需要将输出取反。从而实现同步化和防抖动处理。
如何实现倒计时	根据输入端来设置计数器的初值。使用一个 1 秒的定时器，倒计时开始时启动定时器，每接到定时器完成脉冲就将计数器减一然后重新启动定时器。当计数器为 0 切定时器完成信

	号为 1 时，就输出完成信号 1，这个完成信号一定也是一个长度为 1 个周期的脉冲。
如何实现儿童锁	使用一个寄存器来作为儿童锁状态的输出。将模式选择按钮和水位选择按钮的原始异步信号取与后传入儿童锁模块，然后进行同步化、防抖动处理，记此时的同步输出信号为锁信号。使用一个 1 秒的定时器，在锁信号上升沿脉冲时启动定时器，接到定时器完成脉冲时将儿童锁状态取反，然后关闭定时器。
如何实现舱门锁	在主状态机中，如果当前状态为暂停或运行时，舱门锁开启，否则关闭。
如何实现防误关机保护	同舱门锁。
如何实现强制断电	将电源按钮原始异步信号进行同步化、防抖动处理。同步后电源按钮信号上升沿脉冲时启动一个 2 秒的定时器，接到定时器完成脉冲时将电源状态设为 0，该电源输出信号即为其余模块的复位信号（低电平有效）。
如何表示模式和当前模式的执行状态	使用 3 位二进制数表示，第 0 位表示是否有洗涤程序，第 1 位表示是否有漂洗程序，第 2 位表示是否有脱水程序。执行时只需要从低到高依次执行设置为 1 的位所代表的程序。具体实现可以用补码表示的二进制数的 low bit 操作实现。
如何表示程序和当前程序的执行状态	使用 5 位二进制数表示，从低位到高位依次表示是否有排水过程、甩干过程、进水过程、洗衣过程、漂衣过程。其余规则和原理与模式的表示相同。

3 总结与心得

3.1 课设总结

拿到基本需求后，最初是直接开始编写代码，发现代码总是要来回参考、反复小修小改，而且综合设计时有很多警告，效率低下又容易出错。于是花了一天左右在网上学习 Verilog HDL 的编写规范和部分常用结构的编写方法和思路。然后重新开始本次课程设计，为了高效实现洗衣机控制器的功能及自定义规则要求，接下来做了如下几点工作：

（1）确定自定义规则以及添加的功能：儿童锁、防误关机、强制断电、舱门锁。

（2）确定系统输入输出端口，参见表 2-1、2-2。编写约束文件。

（3）参数传递约定：所有模块所需要的时间参数均通过 parameter 来传递，方便仿真和调节。

（4）各模块接口约定：所有模块均接受系统时钟输入；除电源模块外，所有模块都接受电源信号的输入，电源信号也就是各触发器的复位信号（低电平有效）。

（5）完成串行任务的模块接口约定：所有用来完成串行任务的模块均提供完成脉冲输出（done），完成脉冲是一个长度为一个系统时钟周期的 1 信号，表示该模块负责的任务已经结束，该模块的上级模块在接到该完成脉冲后负责重置该模块。

（6）根据需求，除顶层模块外，将其余模块按照输入、输出、控制、业务、工具分为五类，如下所示：

a. 输入模块：按钮。

b. 输出模块：LED、蜂鸣器、数码管。

c. 控制模块：电源、儿童锁。

d. 业务模块：主状态机、设置、模式运行、程序运行。

e. 工具模块：定时器、嵌套定时器、分频器、防抖动。

（7）从底向上依次实现各模块（包括顶层模块，共 15 个），并对逻辑容易出错而且不容易上板测试的模块进行仿真（防抖动模块、蜂鸣器模块）。期间遇到的问题参见表 2-3。

（8）调整顶层模块的时间参数，对顶层模块进行仿真、调试并修改相关代码，最终做到综合设计和实现设计不报任何警告。

（9）整理所有代码，清除调试和其他不必要的代码，统一代码风格、规范。

（10）下板测试，符合原始规则和自定义规则，效果让本人感到满意。

3.2 课设心得

（1）在计算机程序设计方面上的经验可以大量地用在这次课设上，这次课设最终实现的基本思路是事件驱动。得益于关于接口和脉冲的约定，可以高效而正确地实现各个模块，同时保持了良好的可扩展性。如果有其他需求，可以在较短周期内加入到现有系统中。

（2）更深一步地理解了 Verilog HDL，明确了各个类型的区别、定义，学习了设计同步时序逻辑电路的基本方法（包括组合逻辑描述、状态机描述等）。确定了使用 Verilog HDL 进行编写的习惯、代码风格、规范。

（3）在课设开始的第一天走的弯路没有白走。在之后查阅资料的时候，在一定程度上了解了工业级代码的编写方式，看到了很多小模块的实现方法和思路，让我受益颇深，并将一部分应用到了本次课设中。

（4）虽然每日早起晚归让我每晚都感到疲惫不堪，但是学到了很多知识、积累了很多经验，又加深了对并行化和结构化程序设计的理解。非常感谢有这个机会来让我学习到这么多知识。

4 参考文献

[1] IEEE Std 1364-2005, IEEE Standard for Verilog® Hardware Description Language[S].

[2] Forum for Electronics[EB/OL]. http://www.edaboard.com/.

[3] Xilinx. Vivado Design Suite User Guide-Synthesis[EB/OL]. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug901-vivado-synthesis.pdf.

[4] DIGILENT. Nexys 4 DDR Reference Manual[EB/OL]. https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual.

[5] 欧阳星明，于俊清. 数字逻辑[Z]. 武汉：华中科技大学出版社,2012.

附　录 1（源程序）

button.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module button(tr_btn, a_btn, lock, clk, rst_n);
    parameter DEB_CMAX = `c_ms(5);
    output tr_btn;
    input a_btn;
    input lock, clk, rst_n;

    reg r_lock = 'b0;
    wire pe_btn, ne_btn;
    assign tr_btn = ne_btn && !r_lock && !lock;
    always @(posedge clk)
        if (!rst_n)
            r_lock <= 'b0;
        else if (pe_btn)
            r_lock <= lock;
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_btn(
        .sig(),
        .pe_sig(pe_btn),
        .ne_sig(ne_btn),
        .a_sig(a_btn),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

buzzer.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module buzzer(buz, tr_a, tr_b, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_a, tr_b;
    input clk, rst_n;

    wire buz_a, buz_b;
    assign buz = buz_a || buz_b;
    _buz_a #(
```

```
        .FLA_CMAX(FLA_CMAX)
    ) x_buz_a(
        .buz(buz_a),
        .tr_sig(tr_a),
        .clk(clk),
        .rst_n(rst_n)
    );
    _buz_b #(
        .FLA_CMAX(FLA_CMAX)
    ) x_buz_b(
        .buz(buz_b),
        .tr_sig(tr_b),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule

module _buz_a(buz, tr_sig, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_sig;
    input clk, rst_n;

    //    0   1   0
    //    >   +   -
    reg st = 'd0;
    wire tm_done;
    wire tm_clr = tr_sig;
    assign buz = st;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            st <= 'd0;
        else if (tr_sig)
            st <= 'd1;
        else if (tm_done && st)
            st <= 'd0;
    timer #(
        .CMAX(FLA_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule

module _buz_b(buz, tr_sig, clk, rst_n);
    parameter FLA_CMAX = `c_ms(100);
    output buz;
    input tr_sig;
    input clk, rst_n;
```

```
    //    0   1   2   3   4   5   6   7   8   9 10 11 12 13 14 15 16 17 18 19 20 21   0
    //    >   +   -   +   -   +   -   -   -   +   -   +   -   +   -   -   -   +   -   +   -   +   -
    reg [4:0] st = 'd0;
    wire tm_done;
    wire tm_clr = tr_sig;
    assign buz = st[0] && st != 7 && st != 15;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            st <= 'd0;
        else if (tr_sig)
            st <= 'd1;
        else if (tm_done && st)
            st <= st == 'd21 ? 'd0 : st + 'd1;
    timer #(
        .CMAX(FLA_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

debouncer.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module debouncer(sig, pe_sig, ne_sig, a_sig, clk, rst_n);
    parameter DEB_CMAX = `c_ms(5);
    output reg sig = 'b0;
    output pe_sig, ne_sig;
    input a_sig;
    input clk, rst_n;

    reg r_a_sig = 'b0;
    wire tm_done;
    wire tm_clr = sig == r_a_sig;
    // reg sig
    assign pe_sig = tm_done && !sig;
    assign ne_sig = tm_done && sig;
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            sig <= 'b0;
            r_a_sig <= 'b0;
        end
        else begin
            r_a_sig <= a_sig;
            if (tm_done)
                sig <= !sig;
```

```
        end
    timer #(
        .CMAX(DEB_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

display.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module display(seg_n, an_n, u_tot, u_cur, u_wat, fl_disp, clk_fl, clk, rst_n);
    parameter SCA_CMAX = `c_ms(1);
    output [7:0] seg_n, an_n;
    input [5:0] u_tot, u_cur, u_wat;
    input fl_disp;
    input clk_fl, clk, rst_n;

    wire clk_sca;
    wire [3:0] mem[7:0];
    assign mem[3] = 'b1111;
    assign mem[2] = 'b1111;
    wire [2:0] pos;
    // fwd seg_n
    assign an_n = ~((rst_n && (!fl_disp || clk_fl)) << pos);
    _disp_decimal x_dec_tot(
        .e1(mem[7]),
        .e0(mem[6]),
        .val(u_tot)
    );
    _disp_decimal x_dec_cur(
        .e1(mem[5]),
        .e0(mem[4]),
        .val(u_cur)
    );
    _disp_decimal x_dec_wat(
        .e1(mem[1]),
        .e0(mem[0]),
        .val(u_wat)
    );
    divider #(
        .CMAX(SCA_CMAX)
    ) x_divider(
        .clk_div(clk_sca),
        .clk(clk),
```

```
        .rst_n(rst_n)
    );
    _disp_counter8 x_counter(
        .cnt(pos),
        .clk(clk_sca),
        .rst_n(rst_n)
    );
    _disp_pattern x_pattern(
        .seg_n(seg_n),
        .val(mem[pos])
    );
endmodule

module _disp_decimal(e1, e0, val);
    output [3:0] e1, e0;
    input [5:0] val;
    assign e1 = val / 10;
    assign e0 = val % 10;
endmodule

module _disp_counter8(cnt, clk, rst_n);
    output reg [2:0] cnt = 'd0;
    input clk, rst_n;

    // reg cnt
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            cnt <= 'd0;
        else
            cnt <= cnt + 'd1;
endmodule

module _disp_pattern(seg_n, val);
    output reg [7:0] seg_n; // combinational
    input [3:0] val;
    always @(*)
        case (val)
            'd0:        seg_n <= 'b11000000;
            'd1:        seg_n <= 'b11111001;
            'd2:        seg_n <= 'b10100100;
            'd3:        seg_n <= 'b10110000;
            'd4:        seg_n <= 'b10011001;
            'd5:        seg_n <= 'b10010010;
            'd6:        seg_n <= 'b10000010;
            'd7:        seg_n <= 'b11111000;
            'd8:        seg_n <= 'b10000000;
            'd9:        seg_n <= 'b10010000;
            default: seg_n <= 'b11111111;
        endcase
endmodule
```

divider.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module divider(clk_div, clk, rst_n);
`include "h_cbit.v"
	parameter CMAX = `c_ms(500);
	localparam CBIT = cbit(CMAX);
	output reg clk_div = 'b1;
	input clk, rst_n;

	reg [CBIT - 1:0] cnt = 'd0;
	// reg clk_div
	always @(posedge clk, negedge rst_n)
		if (!rst_n) begin
			clk_div <= 'b1;
			cnt <= 'd0;
		end
		else if (cnt == CMAX) begin
			clk_div <= !clk_div;
			cnt <= 'd1;
		end
		else
			cnt <= cnt + 'd1;
endmodule
```

h_cbit.v

```
function integer cbit(input time cmax);
	begin
		for (cbit = 0; cmax; cbit = cbit + 1)
			cmax = cmax >> 1;
	end
endfunction
```

h_cmax.v

```
`define c_bi(x) (1 << (x))
`define c_cp(x) (x)
`define c_us(x) ((x) * 100)
`define c_ms(x) ((x) * 100_000)
`define c_s(x) ((x) * 100_000_000)
`define c_hz(x) (100_000_000 / (x))
```

h_time.v

```
`define TC_DRA 1
`define TC_SPI 1
`define TC_FIL 1
`define TC_WAS 3
```

```
`define TC_RIN 2
`define TG_WAS (`TC_FIL + `TC_WAS)
`define TG_RIN (`TC_DRA + `TC_SPI + `TC_FIL + `TC_RIN)
`define TG_DRY (`TC_DRA + `TC_SPI)
```

leds.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module leds(
	led_dry, led_rin, led_was,
	led_fil, led_spi, led_dra,
	ld_drw, fl_drw, ld_fsd,
	clk_fl, rst_n
);
	output led_dry, led_rin, led_was;
	output led_fil, led_spi, led_dra;
	input [2:0] ld_drw, fl_drw, ld_fsd;
	input clk_fl, rst_n;

	// fwd led_dry
	// fwd led_rin
	// fwd led_was
	// fwd led_fil
	// fwd led_fil
	// fwd led_spi
	// fwd led_dra
	_led x_led_dry(
		.led(led_dry),
		.ld(ld_drw[2]),
		.fl(fl_drw[2]),
		.clk_fl(clk_fl),
		.rst_n(rst_n)
	);
	_led x_led_rin(
		.led(led_rin),
		.ld(ld_drw[1]),
		.fl(fl_drw[1]),
		.clk_fl(clk_fl),
		.rst_n(rst_n)
	);
	_led x_led_was(
		.led(led_was),
		.ld(ld_drw[0]),
		.fl(fl_drw[0]),
		.clk_fl(clk_fl),
		.rst_n(rst_n)
	);
	_led x_led_fil(
```

```
        .led(led_fil),
        .ld(ld_fsd[2]),
        .fl('b0),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_spi(
        .led(led_spi),
        .ld(ld_fsd[1]),
        .fl('b0),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    _led x_led_dra(
        .led(led_dra),
        .ld(ld_fsd[0]),
        .fl('b0),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
endmodule

module _led(led, ld, fl, clk_fl, rst_n);
    output led;
    input ld, fl;
    input clk_fl, rst_n;

    assign led = rst_n && (fl ? clk_fl : ld);
endmodule
```

main.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module main(
    done, prog_done,
    ioh,
    ld_drw, fl_drw, ld_fsd,
    u_tot, u_cur, u_wat, fl_disp,
    tr_pwr, tr_mod, tr_run, tr_wat,
    clk, rst_n
);
    parameter TIM_CMAX = `c_ms(1000);
    parameter END_CMAX = `c_s(10);
    output done, prog_done;
    output ioh;
    output [2:0] ld_drw, fl_drw;
    output [2:0] ld_fsd;
    output [5:0] u_tot, u_cur, u_wat;
```

```
output fl_disp;
input tr_pwr, tr_mod, tr_run, tr_wat;
input clk, rst_n;

wire tr_set = tr_mod || tr_wat;

reg [1:0] st = 'b00;
wire started = st[0];
wire se_done;
wire [2:0] se_ld_drw;
wire [5:0] se_u_tot, se_u_cur, se_u_wat;
wire [2:0] se_mode;
wire se_clr = started;
wire [2:0] ru_ld_drw, ru_fl_drw;
wire [2:0] ru_ld_fsd;
wire [5:0] ru_u_tot, ru_u_cur;
wire ru_done;
wire ru_pau = st == 'b11;
wire ru_clr = !started;
wire tm_done;
wire tm_clr = st != 'b10;
assign done = rst_n && ((st == 'b10 && tm_done) || (!started && tr_pwr));
// fwd prog_done
assign ioh = started;
assign ld_drw = started ? ru_ld_drw : se_ld_drw;
assign fl_drw = started ? ru_fl_drw : 'b000;
assign ld_fsd = started ? ru_ld_fsd : 'b000;
assign u_tot = started ? ru_u_tot : se_u_tot;
assign u_cur = started ? ru_u_cur : se_u_cur;
assign u_wat = se_u_wat;
assign fl_disp = ru_pau;
always @(posedge clk, negedge rst_n)
    if (!rst_n)
        st <= 'b00;
    else case (st)
        'b00: // setting
            if (se_done)
                st <= 'b01;
        'b01: // running
            if (ru_done)
                st <= 'b10;
            else if (tr_run)
                st <= 'b11;
        'b10: // waiting_end
            if (tr_set)
                st <= 'b00;
            else if (se_done)
                st <= 'b01;
        'b11: // paused
            if (tr_run)
                st <= 'b01;
            else if (tr_set)
```

```
                        st <= 'b00;
            endcase
        setting x_setting(
            .done(se_done),
            .mode(se_mode),
            .ld_drw(se_ld_drw),
            .u_tot(se_u_tot),
            .u_cur(se_u_cur),
            .u_wat(se_u_wat),
            .tr_mod(tr_mod),
            .tr_run(tr_run),
            .tr_wat(tr_wat),
            .clr(se_clr),
            .clk(clk),
            .rst_n(rst_n)
        );
        run_mode #(
            .TIM_CMAX(TIM_CMAX)
        ) x_run_mode(
            .done(ru_done),
            .prog_done(prog_done),
            .ld_drw(ru_ld_drw),
            .fl_drw(ru_fl_drw),
            .ld_fsd(ru_ld_fsd),
            .u_tot(ru_u_tot),
            .u_cur(ru_u_cur),
            .u_wat(se_u_wat),
            .init(se_mode),
            .pau(ru_pau),
            .clr(ru_clr),
            .clk(clk),
            .rst_n(rst_n)
        );
        timer #(
            .CMAX(END_CMAX)
        ) x_timer(
            .done(tm_done),
            .clr(tm_clr),
            .clk(clk),
            .rst_n(rst_n)
        );
    endmodule
```

power.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module power(rst_n, tr_pwr, led_pwr, a_pwr, main_done, lock, clk);
    parameter TIM_CMAX = `c_ms(2000);
```

```
    parameter DEB_CMAX = `c_ms(5);
    output reg rst_n = 'b0;
    output tr_pwr;
    output led_pwr;
    input a_pwr;
    input main_done;
    input lock, clk;

    // power button with lock
    reg r_lock = 'b0;
    wire tm_done;
    wire tm_clr = !pwr;
    // forcing poweroff
    reg r_rst_n = 'b0;
    wire pwr, pe_pwr, ne_pwr;
    // reg rst_n
    assign tr_pwr = ne_pwr && !r_lock && !lock && r_rst_n == rst_n;
    assign led_pwr = rst_n;
    always @(posedge clk)
        if (tm_done || main_done)
            rst_n <= 'b0;
        else if (!rst_n && tr_pwr)
            rst_n <= 'b1;
        else if (pe_pwr)
            r_rst_n <= rst_n;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            r_lock <= 'b0;
        else if (pe_pwr)
            r_lock <= lock;
    timer #(
        .CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_pwr(
        .sig(pwr),
        .pe_sig(pe_pwr),
        .ne_sig(ne_pwr),
        .a_sig(a_pwr),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

protector.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module protector(lock, tr_lck, led_lck, a_lck, clk, rst_n);
    parameter TIM_CMAX = `c_ms(1000);
    parameter DEB_CMAX = `c_ms(5);
    output reg lock = 'b0;
    output tr_lck;
    output led_lck;
    input a_lck;
    input clk, rst_n;

    reg r_lock = 'b0;
    wire tm_done;
    wire lck, pe_lck;
    wire tm_clr = !lck || r_lock != lock;
    // reg lock
    assign tr_lck = tm_done;
    assign led_lck = lock;
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            lock <= 'b0;
            r_lock <= 'b0;
        end
        else if (tm_done)
            lock <= !lock;
        else if (pe_lck)
            r_lock <= lock;

    timer #(
        .CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
    debouncer #(
        .DEB_CMAX(DEB_CMAX)
    ) x_deb_lck(
        .sig(lck),
        .pe_sig(pe_lck),
        .ne_sig(),
        .a_sig(a_lck),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

run_mode.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"
`include "h_time.v"

module run_mode(
	done, prog_done,
	ld_drw, fl_drw, ld_fsd,
	u_tot, u_cur,
	u_wat, init, pau, clr, clk, rst_n
);
	parameter TIM_CMAX = `c_ms(1000);
	output done, prog_done;
	output [2:0] ld_drw, fl_drw;
	output [2:0] ld_fsd;
	output [5:0] u_tot, u_cur;
	input [5:0] u_wat;
	input [2:0] init;
	input pau, clr, clk, rst_n;

	reg [2:0] st = 'b000;
	wire [2:0] lowb = st & -st;
	wire [2:0] srem = st ^ lowb;
	reg r_prog_done = 'b0;
	reg [4:0] prog_init; // combinational
	always @(*)
		case (lowb)
			'b001:	prog_init <= 'b01100;
			'b010:	prog_init <= 'b10111;
			'b100:	prog_init <= 'b00011;
			default: prog_init <= 'b00000;
		endcase
	reg r_clr = 'b0;
	wire prog_clr = r_clr || r_prog_done;
	assign done = !st && r_prog_done;
	// fwd prog_done
	assign ld_drw = st;
	assign fl_drw = lowb;
	// fwd ld_fsd
	assign u_tot = u_cur + u_wat * (
		(srem[0] ? `TG_WAS : 0) +
		(srem[1] ? `TG_RIN : 0) +
		(srem[2] ? `TG_DRY : 0)
	);
	// fwd u_cur
	always @(posedge clk, negedge rst_n)
		if (!rst_n) begin
			st <= 'b000;
			r_prog_done <= 'b0;
			r_clr <= 'b0;
		end
```

```
        else if (!pau) begin
            r_prog_done <= prog_done;
            r_clr <= clr;
            if (clr) begin
                st <= init;
            end
            else if (prog_done)
                st <= srem;
        end
    run_prog #(
        .TIM_CMAX(TIM_CMAX)
    ) x_run_prog(
        .done(prog_done),
        .ld_fsd(ld_fsd),
        .u_cur(u_cur),
        .u_wat(u_wat),
        .init(prog_init),
        .pau(pau),
        .clr(prog_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

run_prog.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"
`include "h_time.v"

module run_prog(
    done,
    ld_fsd, u_cur,
    u_wat, init, pau, clr, clk, rst_n
);
    parameter TIM_CMAX = `c_ms(1000);
    output done;
    output [2:0] ld_fsd;
    output [5:0] u_cur;
    input [5:0] u_wat;
    input [4:0] init;
    input pau, clr, clk, rst_n;

    reg[4:0] st = 'b00000;
    wire [4:0] lowb = st & -st;
    wire [4:0] srem = st ^ lowb;
    reg [5:0] tm_init_mul; // combinational
    always @(*)
        case (lowb)
            'b00001: tm_init_mul <= `TC_DRA;
```

```
                'b00010: tm_init_mul <= `TC_SPI;
                'b00100: tm_init_mul <= `TC_FIL;
                'b01000: tm_init_mul <= `TC_WAS;
                'b10000: tm_init_mul <= `TC_RIN;
                default: tm_init_mul <= 'd0;
        endcase
    wire tm_done;
    reg r_tm_done = 'b0;
    wire [5:0] tm_rema;
    wire [5:0] tm_init = u_wat * tm_init_mul;
    reg r_clr = 'b0;
    wire tm_clr = r_clr || r_tm_done;
    assign done = !st && r_tm_done;
    assign ld_fsd = lowb[2:0];
    assign u_cur = tm_rema + u_wat * (
        (srem[0] ? `TC_DRA : 0) +
        (srem[1] ? `TC_SPI : 0) +
        (srem[2] ? `TC_FIL : 0) +
        (srem[3] ? `TC_WAS : 0) +
        (srem[4] ? `TC_RIN : 0)
    );
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            st <= 'b00000;
            r_tm_done <= 'b0;
            r_clr <= 'b0;
        end
        else if (!pau) begin
            r_tm_done <= tm_done;
            r_clr <= clr;
            if (clr)
                st <= init;
            else if (tm_done)
                st <= srem;
        end
    timer_nested #(
        .CBIT(6),
        .INNER_CMAX(TIM_CMAX)
    ) x_timer(
        .done(tm_done),
        .rema(tm_rema),
        .init(tm_init),
        .pau(pau),
        .clr(tm_clr),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

setting.v

```
`timescale 1ns / 1ps

`include "h_time.v"

module setting(
    done, mode,
    ld_drw,
    u_tot, u_cur, u_wat,
    tr_mod, tr_run, tr_wat,
    clr, clk, rst_n
);
    output done;
    output reg [2:0] mode = 'b111;
    output [2:0] ld_drw;
    output [5:0] u_tot, u_cur;
    output reg [5:0] u_wat = 'd3;
    input tr_mod, tr_run, tr_wat;
    input clr, clk, rst_n;

    // drw =>    w =>   rw =>   r   => dr   => d     => drw
    // 111       001      011      010      110      100      111
    assign done = tr_run;
    // reg mode
    assign ld_drw = mode;
    assign u_tot = u_wat * (
        (mode[0] ? `TG_WAS : 0) +
        (mode[1] ? `TG_RIN : 0) +
        (mode[2] ? `TG_DRY : 0)
    );
    assign u_cur = u_wat * (
        mode[0] ? `TG_WAS :
        mode[1] ? `TG_RIN :
        mode[2] ? `TG_DRY : 0
    );
    // reg u_wat
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            mode <= 'b111;
            u_wat <= 'd3;
        end
        else if (clr)
            mode <= 'b111;
        else if (tr_mod)
            case (mode)
                'b111:    mode <= 'b001;
                'b001:    mode <= 'b011;
                'b011:    mode <= 'b010;
                'b010:    mode <= 'b110;
                'b110:    mode <= 'b100;
                'b100:    mode <= 'b111;
                default: mode <= 'b111;
            endcase
```

```
        else if (tr_wat)
            u_wat <= u_wat == 'd5 ? 'd2 : u_wat + 'd1;
endmodule
```

timer.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module timer(done, clr, clk, rst_n);
`include "h_cbit.v"
    parameter CMAX = `c_ms(1000);
    localparam CBIT = cbit(CMAX);
    output done;
    input clr, clk, rst_n;

    reg [CBIT - 1:0] cnt = 'd0;
    assign done = cnt == CMAX;
    always @(posedge clk, negedge rst_n)
        if (!rst_n)
            cnt <= 'd0;
        else if (clr)
            cnt <= 'd0;
        else
            cnt <= done ? 'd1 : cnt + 'd1;
endmodule
```

timer_nested.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module timer_nested(done, rema, init, pau, clr, clk, rst_n);
    parameter CBIT = 6;
    parameter INNER_CMAX = `c_ms(1000);
    output done;
    output reg [CBIT - 1:0] rema = 'd0;
    input [CBIT - 1:0] init;
    input pau, clr, clk, rst_n;

    wire ok = !rema;
    wire tm_done;
    reg r_tm_done = 'b0;
    assign done = r_tm_done && ok;
    // reg rema
    always @(posedge clk, negedge rst_n)
        if (!rst_n) begin
            rema <= 'd0;
            r_tm_done <= 'b0;
```

```
            end
            else if (!pau) begin
                r_tm_done <= tm_done;
                if (clr)
                    rema <= init;
                else if (tm_done)
                    rema <= (ok ? init : rema) - 1;
            end
        timer #(
            .CMAX(INNER_CMAX)
        ) x_timer(
            .done(tm_done),
            .clr(clr),
            .clk(clk),
            .rst_n(rst_n)
        );
endmodule
```

top.v

```
`timescale 1ns / 1ps

`include "h_cmax.v"

module top(
    buz, ioh,
    led_pwr, led_lck,
    led_dry, led_rin, led_was,
    led_fil, led_spi, led_dra,
    seg_n, an_n,
    a_mod, a_run, a_wat, a_pwr,
    clk
);
    parameter TIM_UNIT = `c_ms(1000);
    parameter END_WAIT = `c_s(10);
    parameter DEB_WAIT = `c_ms(5);
    parameter LCK_WAIT = `c_ms(1000);
    parameter FPO_WAIT = `c_ms(2000);
    parameter BUZ_INTV = `c_ms(100);
    parameter FLA_INTV = `c_ms(500);
    parameter DIS_INTV = `c_ms(1);
    output buz, ioh;
    output led_pwr, led_lck;
    output led_dry, led_rin, led_was;
    output led_fil, led_spi, led_dra;
    output [7:0] seg_n, an_n;
    input a_mod, a_run, a_wat, a_pwr;
    input clk;

    wire a_lck = a_mod && a_wat;
```

```
wire clk_fl;
wire [2:0] ld_drw, fl_drw, ld_fsd;
wire [5:0] u_tot, u_cur, u_wat;
wire fl_disp;
wire main_done, prog_done;
wire rst_n, tr_pwr;
wire lock;
wire tr_lck, tr_mod, tr_run, tr_wat;
wire buz_tr_a = tr_lck || tr_mod || tr_run || tr_wat;
wire buz_tr_b = prog_done;
// fwd buz
// fwd led_pwr
// fwd led_lck
// fwd led_dry
// fwd led_rin
// fwd led_was
// fwd led_fil
// fwd led_spi
// fwd led_spi
// fwd led_dra
// fwd seg_n
// fwd an_n
main #(
	.TIM_CMAX(TIM_UNIT),
	.END_CMAX(END_WAIT)
) x_main(
	.done(main_done),
	.prog_done(prog_done),
	.ioh(ioh),
	.ld_drw(ld_drw),
	.fl_drw(fl_drw),
	.ld_fsd(ld_fsd),
	.u_tot(u_tot),
	.u_cur(u_cur),
	.u_wat(u_wat),
	.fl_disp(fl_disp),
	.tr_pwr(tr_pwr),
	.tr_mod(tr_mod),
	.tr_run(tr_run),
	.tr_wat(tr_wat),
	.clk(clk),
	.rst_n(rst_n)
);
power #(
	.TIM_CMAX(FPO_WAIT),
	.DEB_CMAX(DEB_WAIT)
) x_power(
	.rst_n(rst_n),
	.tr_pwr(tr_pwr),
	.led_pwr(led_pwr),
	.a_pwr(a_pwr),
	.main_done(main_done),
```

```
        .lock(lock),
        .clk(clk)
    );
    protector #(
        .TIM_CMAX(LCK_WAIT),
        .DEB_CMAX(DEB_WAIT)
    ) x_protector(
        .lock(lock),
        .tr_lck(tr_lck),
        .led_lck(led_lck),
        .a_lck(a_lck),
        .clk(clk),
        .rst_n(rst_n)
    );
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_mod(
        .tr_btn(tr_mod),
        .a_btn(a_mod),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_run(
        .tr_btn(tr_run),
        .a_btn(a_run),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    button #(
        .DEB_CMAX(DEB_WAIT)
    ) x_deb_wat(
        .tr_btn(tr_wat),
        .a_btn(a_wat),
        .lock(lock),
        .clk(clk),
        .rst_n(rst_n)
    );
    buzzer #(
        .FLA_CMAX(BUZ_INTV)
    ) x_buzzer(
        .buz(buz),
        .tr_a(buz_tr_a),
        .tr_b(buz_tr_b),
        .clk(clk),
        .rst_n(rst_n)
    );
    divider #(
        .CMAX(FLA_INTV)
```

```
    ) x_divider_fl(
        .clk_div(clk_fl),
        .clk(clk),
        .rst_n(rst_n)
    );
    leds x_leds(
        .led_dry(led_dry),
        .led_rin(led_rin),
        .led_was(led_was),
        .led_fil(led_fil),
        .led_spi(led_spi),
        .led_dra(led_dra),
        .ld_drw(ld_drw),
        .fl_drw(fl_drw),
        .ld_fsd(ld_fsd),
        .clk_fl(clk_fl),
        .rst_n(rst_n)
    );
    display #(
        .SCA_CMAX(DIS_INTV)
    ) x_display(
        .seg_n(seg_n),
        .an_n(an_n),
        .u_tot(u_tot),
        .u_cur(u_cur),
        .u_wat(u_wat),
        .fl_disp(fl_disp),
        .clk_fl(clk_fl),
        .clk(clk),
        .rst_n(rst_n)
    );
endmodule
```

constr_top.xdc

```
# Clock
set_property -dict { PACKAGE_PIN E3       IOSTANDARD LVCMOS33 } [get_ports { clk }];
# CLK100MHZ
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];

# LEDs
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { led_pwr }];
# LD0
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports { led_lck }];    #
LD1
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports { led_fil }];    #
LD3
set_property -dict { PACKAGE_PIN R18      IOSTANDARD LVCMOS33 } [get_ports { led_spi }];    #
LD4
set_property -dict { PACKAGE_PIN V17      IOSTANDARD LVCMOS33 } [get_ports { led_dra }];
# LD5
```

```
set_property -dict { PACKAGE_PIN U16     IOSTANDARD LVCMOS33 } [get_ports { led_dry }];
# LD7
set_property -dict { PACKAGE_PIN V16     IOSTANDARD LVCMOS33 } [get_ports { led_rin }];   #
LD8
set_property -dict { PACKAGE_PIN T15     IOSTANDARD LVCMOS33 } [get_ports { led_was }];
# LD9
set_property -dict { PACKAGE_PIN V12     IOSTANDARD LVCMOS33 } [get_ports { ioh }];
# LD14
set_property -dict { PACKAGE_PIN V11     IOSTANDARD LVCMOS33 } [get_ports { buz }];
# LD15

# 7 segment display
set_property -dict { PACKAGE_PIN T10     IOSTANDARD LVCMOS33 } [get_ports { seg_n[0] }]; #
CA
set_property -dict { PACKAGE_PIN R10     IOSTANDARD LVCMOS33 } [get_ports { seg_n[1] }]; #
CB
set_property -dict { PACKAGE_PIN K16     IOSTANDARD LVCMOS33 } [get_ports { seg_n[2] }]; #
CC
set_property -dict { PACKAGE_PIN K13     IOSTANDARD LVCMOS33 } [get_ports { seg_n[3] }]; #
CD
set_property -dict { PACKAGE_PIN P15     IOSTANDARD LVCMOS33 } [get_ports { seg_n[4] }]; #
CE
set_property -dict { PACKAGE_PIN T11     IOSTANDARD LVCMOS33 } [get_ports { seg_n[5] }]; #
CF
set_property -dict { PACKAGE_PIN L18     IOSTANDARD LVCMOS33 } [get_ports { seg_n[6] }]; #
CG
set_property -dict { PACKAGE_PIN H15     IOSTANDARD LVCMOS33 } [get_ports { seg_n[7] }]; #
DP
set_property -dict { PACKAGE_PIN J17     IOSTANDARD LVCMOS33 } [get_ports { an_n[0] }];   #
AN0
set_property -dict { PACKAGE_PIN J18     IOSTANDARD LVCMOS33 } [get_ports { an_n[1] }];   #
AN1
set_property -dict { PACKAGE_PIN T9      IOSTANDARD LVCMOS33 } [get_ports { an_n[2] }];
# AN2
set_property -dict { PACKAGE_PIN J14     IOSTANDARD LVCMOS33 } [get_ports { an_n[3] }];   #
AN3
set_property -dict { PACKAGE_PIN P14     IOSTANDARD LVCMOS33 } [get_ports { an_n[4] }];   #
AN4
set_property -dict { PACKAGE_PIN T14     IOSTANDARD LVCMOS33 } [get_ports { an_n[5] }];
# AN5
set_property -dict { PACKAGE_PIN K2      IOSTANDARD LVCMOS33 } [get_ports { an_n[6] }];
# AN6
set_property -dict { PACKAGE_PIN U13     IOSTANDARD LVCMOS33 } [get_ports { an_n[7] }];
# AN7

# Buttons
set_property -dict { PACKAGE_PIN N17     IOSTANDARD LVCMOS33 } [get_ports { a_run }];
# BTNC
set_property -dict { PACKAGE_PIN M18     IOSTANDARD LVCMOS33 } [get_ports { a_pwr }];
# BTNU
set_property -dict { PACKAGE_PIN P17     IOSTANDARD LVCMOS33 } [get_ports { a_mod }];
# BTNL
```

```
set_property -dict { PACKAGE_PIN M17     IOSTANDARD LVCMOS33 } [get_ports { a_wat }];
# BTNR

# Device
set_property CFGBVS VCCO              [current_design];
set_property CONFIG_VOLTAGE 3.3 [current_design];
```

test/tb_buzzer.v

```
`timescale 1ns / 1ps

module tb_buzzer();
`include "tb_h_common.v"
`define press(x) begin `us(2) x = 1; `us(1) x = 0; end
    reg a_a = 0;
    reg a_b = 0;
    initial begin
        `press(a_a);
        `us(10)   `press(a_b);
        `us(4.7) `press(a_a);
        `us(120) `press(a_b);
        `us(12)   `press(a_a);
    end
    wire tr_a, tr_b;
    buzzer #(
        .FLA_CMAX(`c_us(5))
    ) x_dut(
        .tr_a(tr_a),
        .tr_b(tr_b),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_a(
        .tr_btn(tr_a),
        .a_btn(a_a),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_b(
        .tr_btn(tr_b),
        .a_btn(a_b),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

tb_debouncer.v

```
`timescale 1ns / 1ps

module tb_debouncer();
`include "tb_h_common.v"
	reg a_sig = 'b0;
	initial begin
		`cp(1.3)    a_sig = 1;
		`cp(10.1) a_sig = 0;
		`cp(9.9)    a_sig = 1;
		`cp(2.8)    a_sig = 0;
		`cp(10.1) a_sig = 1;
		`cp(9.9)    a_sig = 0;
	end
	debouncer #(
		.DEB_CMAX(`c_cp(10))
	) x_dut(
		.a_sig(a_sig),
		.clk(clk),
		.rst_n('b1)
	);
endmodule
```

tb_h_common.v

```
`include "../h_cmax.v"

`define cp(x) #((x) * 10)
`define ns(x) #(x)
`define us(x) #((x) * 1000)
`define ms(x) #((x) * 1000_000)

	reg clk = 'b1;
	always #5 clk <= !clk;
```

tb_running.v

```
`timescale 1ns / 1ps

module tb_running();
`include "tb_h_common.v"
	reg a_pau = 0;
	reg a_clr = 1;
	reg [2:0] init = 'b111;
	reg [5:0] u_wat = 3;
	initial begin
		`us(1)      a_clr = 0;
		`us(800) a_clr = 1;
		`us(1)      init = 'b011;
		`us(1)      a_clr = 0;
```

```
        `us(82)    a_pau = 1;
        `us(800) a_pau = 0;
    end
    wire pau, clr;
    run_mode #(
        .TIM_CMAX(`c_us(10))
    ) x_dut(
        .u_wat(u_wat),
        .init(init),
        .pau(pau),
        .clr(clr),
        .clk(clk),
        .rst_n('b1)
    );
    debouncer #(
        .DEB_CMAX(`c_cp(10))
    ) x_deb_pau(
        .sig(pau),
        .a_sig(a_pau),
        .clk(clk),
        .rst_n('b1)
    );
    debouncer #(
        .DEB_CMAX(`c_cp(10))
    ) x_deb_clr(
        .sig(clr),
        .a_sig(a_clr),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

tb_setting.v

```
`timescale 1ns / 1ps

module tb_setting();
`include "tb_h_common.v"
`define press(x) begin `us(2) x = 1; `us(1) x = 0; end
    reg a_mod = 0;
    reg a_run = 0;
    reg a_wat = 0;
    reg a_clr = 0;
    initial begin
        `press(a_clr);
        repeat (6) `press(a_mod);
        repeat (4) `press(a_wat);
        `us(10) `press(a_clr);
        `press(a_wat);
        `press(a_mod);
        `us(10) a_clr = 1;
```

```
    end
    wire tr_mod, tr_run, tr_wat, clr;
    setting x_dut(
        .tr_mod(tr_mod),
        .tr_run(tr_run),
        .tr_wat(tr_wat),
        .clr(clr),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_mod(
        .tr_btn(tr_mod),
        .a_btn(a_mod),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_run(
        .tr_btn(tr_run),
        .a_btn(a_run),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
    button #(
        .DEB_CMAX(`c_cp(10))
    ) x_btn_wat(
        .tr_btn(tr_wat),
        .a_btn(a_wat),
        .lock('b0),
        .clk(clk),
        .rst_n('b1)
    );
    debouncer #(
        .DEB_CMAX(`c_cp(10))
    ) x_deb_clr(
        .sig(clr),
        .a_sig(a_clr),
        .clk(clk),
        .rst_n('b1)
    );
endmodule
```

tb_top.v

```
`timescale 1ns / 1ps
```

```
module tb_top();
`include "tb_h_common.v"
`define press(x) begin `us(2) x = 1; `us(1) x = 0; end
    wire pwr;
    reg a_mod = 0;
    reg a_run = 0;
    reg a_wat = 0;
    reg a_pwr = 0;
    initial begin
        `press(a_pwr);
        repeat (6) `press(a_mod);
        repeat (4) `press(a_wat);
        `press(a_run);
        `us(20) a_mod = 1;
        `us(4)    a_wat = 1;
        `us(20) a_wat = 0;
        `us(6)    a_mod = 0;
        `us(20) a_pwr = 1;
        `us(30) a_pwr = 0;
        `us(50) `press(a_pwr);
        `press(a_run);
        `us(40) `press(a_run);
        `us(10) `press(a_pwr);
        `us(10) `press(a_run);
        `us(20) a_mod = 1;
        `us(4)    a_wat = 1;
        `us(12) a_mod = 0;
        `us(2)    a_wat = 0;
        `press(a_pwr);
        `press(a_mod);
        `press(a_run);
        `press(a_wat);
        `us(20) a_mod = 1;
        `us(4)    a_wat = 1;
        `us(12) a_mod = 0;
        `us(4)    a_wat = 0;
        `press(a_run);
        `press(a_pwr);
        repeat (5) `press(a_mod);
        `press(a_run);
    end
    top #(
        .TIM_UNIT(`c_us(10)),
        .END_WAIT(`c_us(100)),
        .DEB_WAIT(`c_cp(5)),
        .LCK_WAIT(`c_us(10)),
        .FPO_WAIT(`c_us(20)),
        .BUZ_INTV(`c_us(1)),
        .FLA_INTV(`c_us(5)),
        .DIS_INTV(`c_cp(1))
    ) x_dut(
        .led_pwr(pwr),
```

```
            .a_mod(a_mod),
            .a_run(a_run),
            .a_wat(a_wat),
            .a_pwr(a_pwr),
            .clk(clk)
        );
endmodule
```