Laboratorio de Sistemas Digitales 2

Paralelo: 103

PROYECTO
Tema: Control de atención a Usuarios

Integrantes:

Ochoa Ochoa Ariana

Realpe Villacís Dario

Profesor:

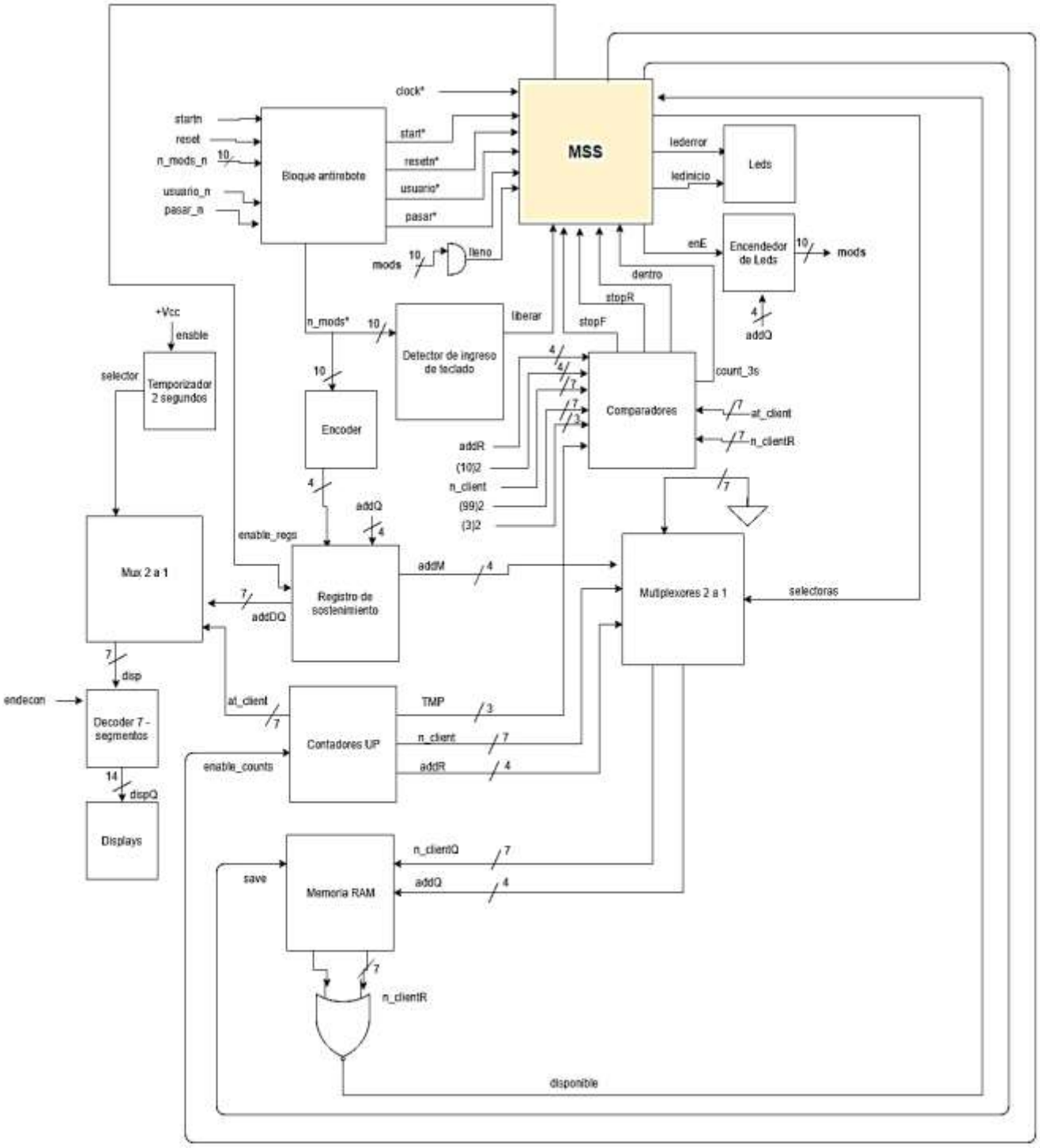
Ing. Sara Ríos

Término:

2018-2019

Centro de Atención a usuarios

1) Diagrama de Bloques 1 (incluye las variables de entrada, salida y señales internas del Sistema)



2) Listado que contenga todas las señales de entrada, salida y señales internas

Señales de entrada

- **start:** permite resetear y habilitar los módulos de atención al usuario, estableciendo el sistema en un estado de activación donde se espera que un usuario pida su ticket con la señal usuario, que se libere algún módulo con la señal liberar o que pase un usuario a su respectivo módulo de atención con la señal pasar.
- **resetn:** Reinicia el sistema digital sin importar en qué estado se encuentre.
- **clock:** señal de reloj que permite los cambios de estados del sistema digital.
- **liberar:** botón que indica que un módulo desea liberar su puesto porque ya atendió a su respectivo cliente, asignándole un nuevo turno a atender siempre que sea posible. Se valida que el módulo de atención este ocupado en verdad y para la asignación de un nuevo turno al mismo que ya no se haya atendido a todos los 99 usuarios de límite.
- **stopF:** indica que se han atendido a 99 usuarios.
- **stopR:** detiene el recorrido de la memoria RAM que contiene el turno que le corresponde a cada módulo de atención.
- **disponible:** indica que un módulo se encuentra vacío.
- **lleno:** indica que todos los módulos están siendo ocupados por un usuario.
- **count_3s:** en caso que se genere un error cuenta tres segundos para encender un indicador led.
- **usuario:** permite darle un turno a un nuevo cliente solo si aún no se han dado 99 turnos y si existe un módulo vacío.
- **dentro:** se activa cuando el número del ticket es el mismo número que ha sido llamado por pantalla en el respectivo módulo de atención.
- **pasar:** indica cuando hay que atender al siguiente cliente; para cumplir con esto se debe verificar que el cliente se dirija al módulo correcto según el número de su ticket.

Señales internas y de salida

- **ledinicio:** señal que activa un diodo led indicando que el sistema se encuentra en el estado de activación, esperando por las señales usuario, liberar y pasar.
- **lederror:** si no se encuentra el ticket asignado en ningún puesto, se da un error, de esta forma se valida que el número de turnos dados sea mayor a los atendidos.
- **enN:** permite habilitar el contador cnt_turno de turnos asignados a los clientes.
- **resetNN:** permite resetear el contador cnt_turno de turnos asignados a los clientes.
- **resetTM:** permite resetear el contador cnt_segL que cuenta 3 segundos para encender el led de error en caso de ser necesario.
- **save:** permite guardar un dato en la memoria RAM ram_mods con la información de los turnos asignados en cada módulo.
- **enR:** permite habilitar el contador cnt_it_mods que permite recorrer la RAM ram_mods.
- **enTM:** permite habilitar el contador cnt_segL.
- **ldR:** permite hacer carga en paralelo al contador cnt_it_mods.
- **select_n:** selectora del mux mux_cliente.
- **select_add:** selectora del mux mux_add.
- **enE:** permite habilitar el encendedor de leds leds_mods, de forma que se encienda el led según la dirección dada en dicho bloque.
- **resetE:** permite resetear el bloque encendedor de leds leds_mods.
- **enAC:** permite habilitar el contador cnt_atendidos con el número de tickets de los clientes que ya han sido atendidos.
- **resetRR:** permite reset el registro de sostenimiento reg_mods, que guarda el número del módulo que ya atendió su ticket.
- **ldAC:** permite hacer carga en paralelo al contador cnt_atendidos.

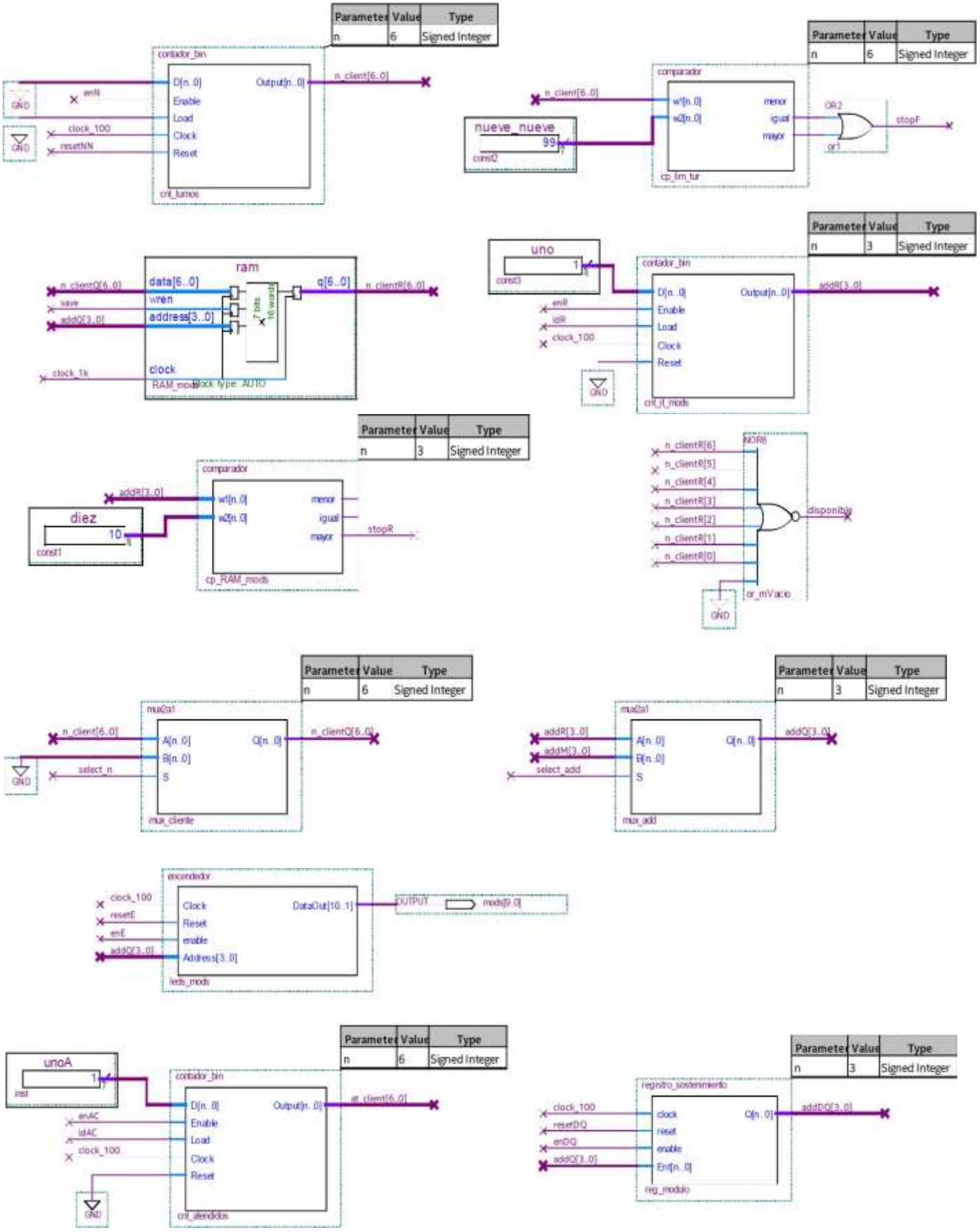
Centro de Atención a usuarios

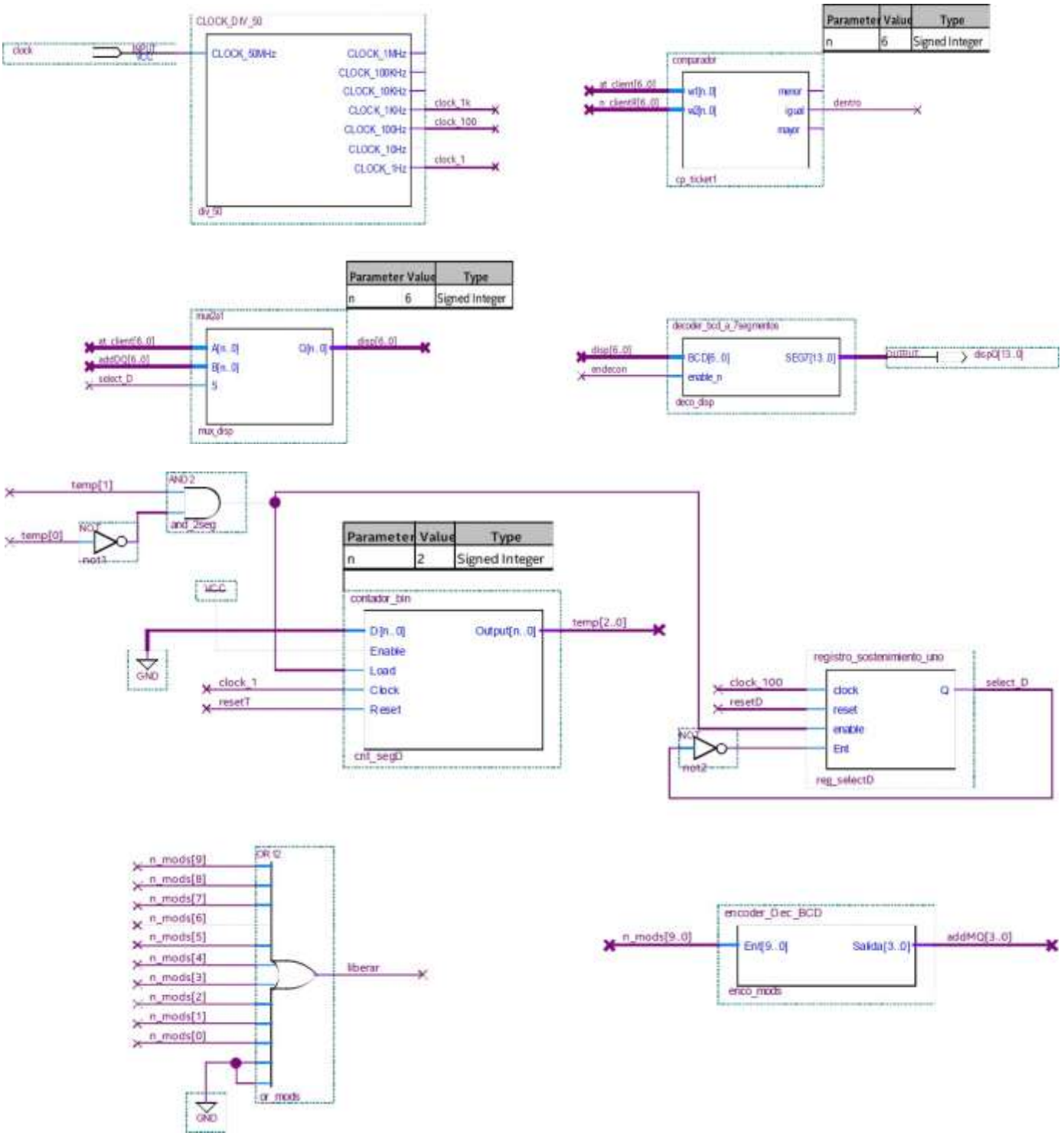
- **enDQ:** permite habilitar el registro de sostenimiento reg_modulo que guarda el módulo de atención que le corresponde al siguiente cliente.
- **resetDQ:** permite resetear el registro de sostenimiento reg_modulo.
- **resetT:** permite resetear el contador cnt_segD que cuenta 2 segundos de forma continua.
- **resetD:** permite resetear el registro de sostenimiento reg_selectD que permite el cambio de la selectora select_D del mux mux_disp para mostrar en los displays el número de turno y módulo que le corresponde al siguiente cliente de forma alternada.
- **enRR:** permite habilitar el registro de sostenimiento reg_mods que guarda el número del módulo que ya atendió su ticket.

Centro de Atención a usuarios

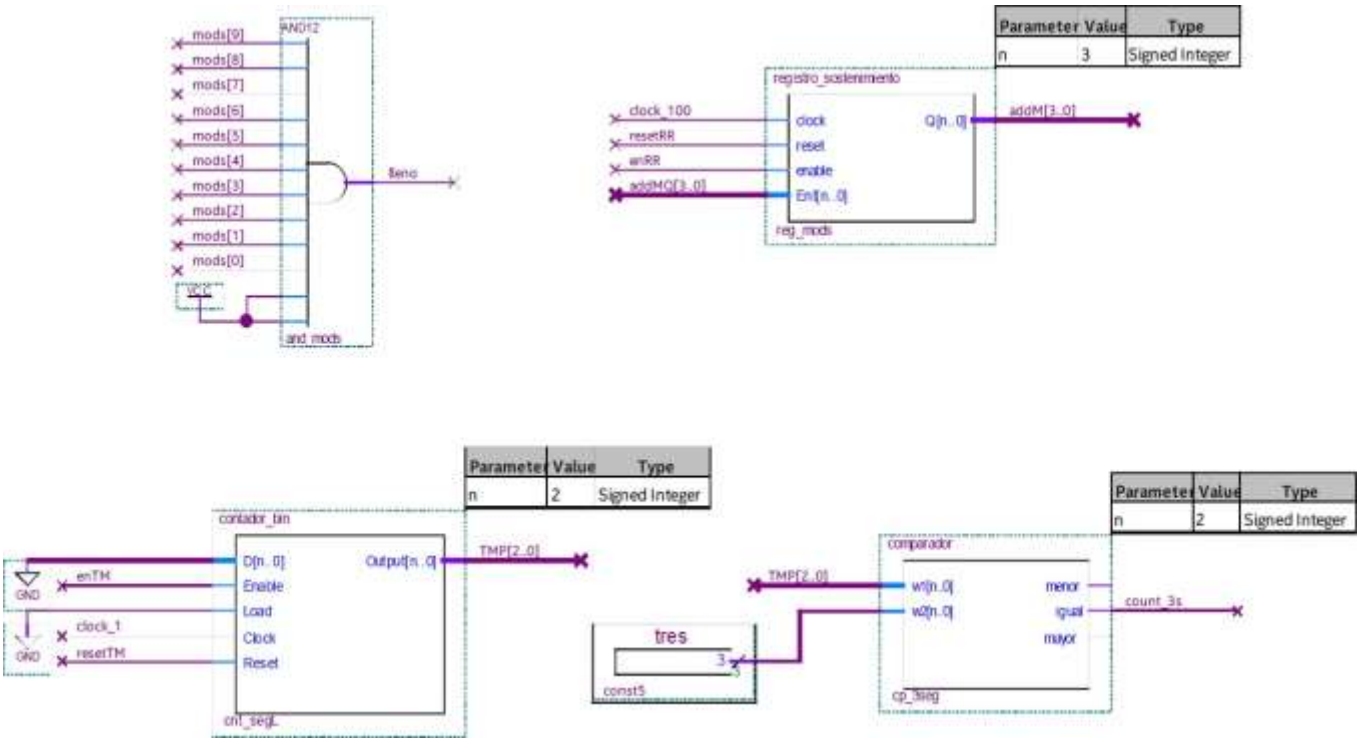
3) Partición Funcional 1

Procesador de datos

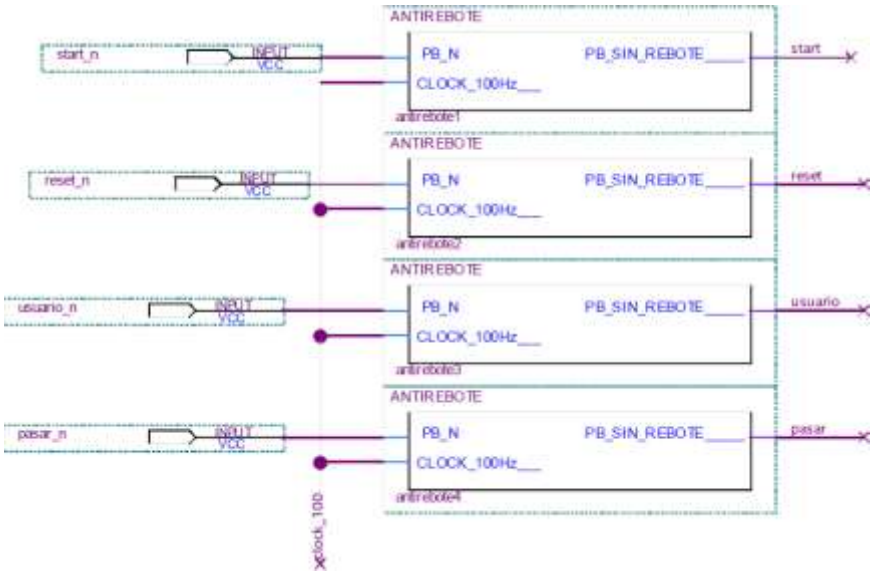



CLOCK_DIV_50
clock
CLOCK_50MHz
CLOCK_1MHz
CLOCK_100KHz
CLOCK_10KHz
CLOCK_1KHz
CLOCK_100Hz
CLOCK_10Hz
CLOCK_1Hz
clock_1k
clock_100
clock_1
div_50
Parameter Value Type
n 6 Signed Integer
comparador
at_client[6..0]
n_client[6..0]
w1[n..0]
w2[n..0]
menor
igual
mayor
dentro
cp_ticket1
Parameter Value Type
n 6 Signed Integer
mux2o1
A[n..0]
B[n..0]
S
O[n..0]
disp[6..0]
addDO[6..0]
select_D
mux_disp
decoder_bcd_a_7segmentos
disp[6..0]
endecon
BCD[6..0]
enable_n
SEG7[13..0]
output
disp0[13..0]
deco_disp
temp[1]
temp[0]
AND2
and_2seg
NOT
not1
VCC
GND
Parameter Value Type
n 2 Signed Integer
contador_bin
D[n..0]
Enable
Load
Clock
Reset
Output[n..0]
temp[2..0]
clock_1
resetT
cnt_segD
registro_sostenimiento_uno
clock_100
resetD
clock
reset
enable
Ent
Q
select_D
not2
reg_selectD
n_mods[9]
n_mods[8]
n_mods[7]
n_mods[6]
n_mods[5]
n_mods[4]
n_mods[3]
n_mods[2]
n_mods[1]
n_mods[0]
OR 10
liberar
or_mods
GND
encoder_Dec_BCD
n_mods[9..0]
En[9..0]
Salida[3..0]
addMO[3..0]
enco_mods

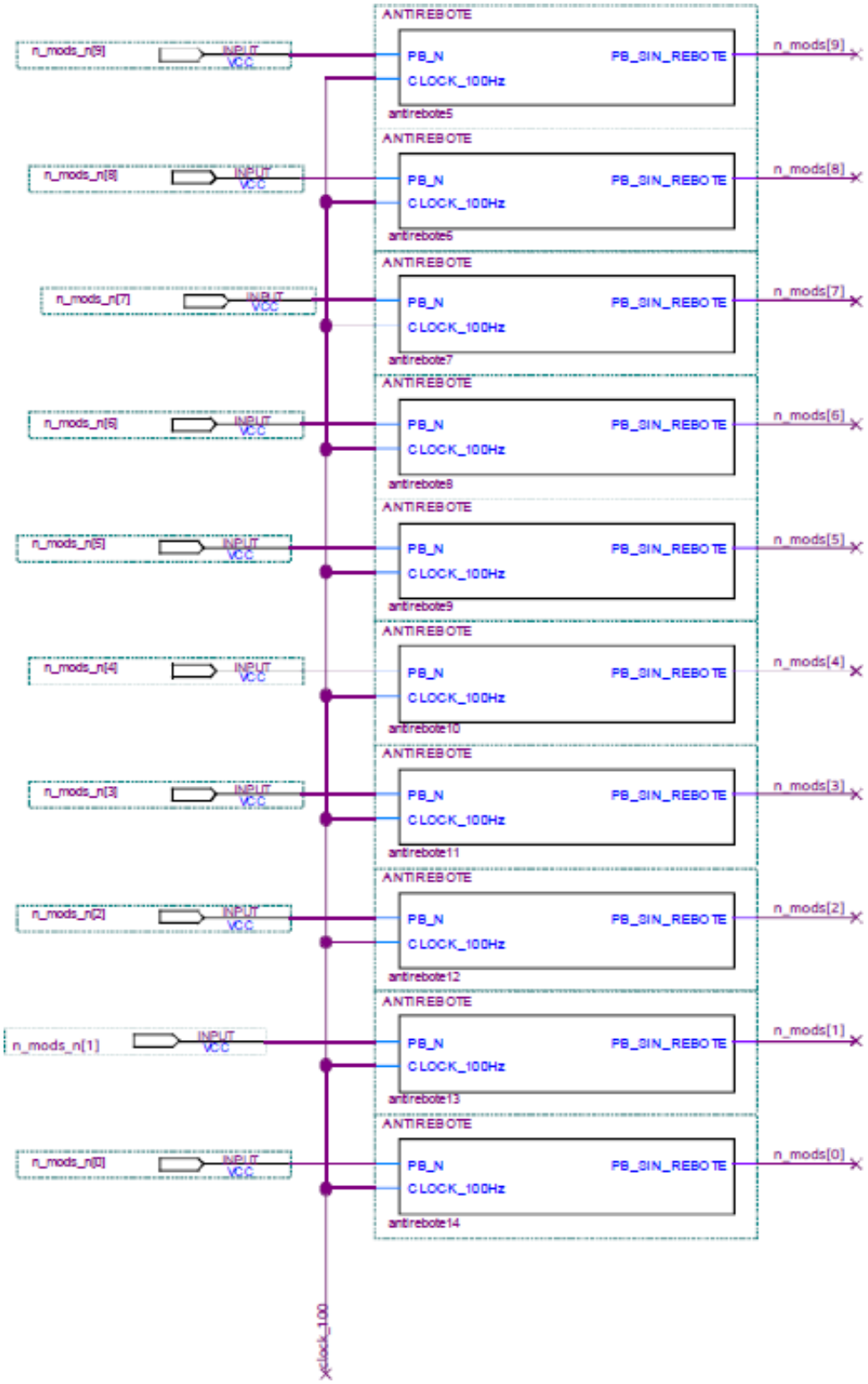
Centro de Atención a usuarios



Bloques anti-rebote

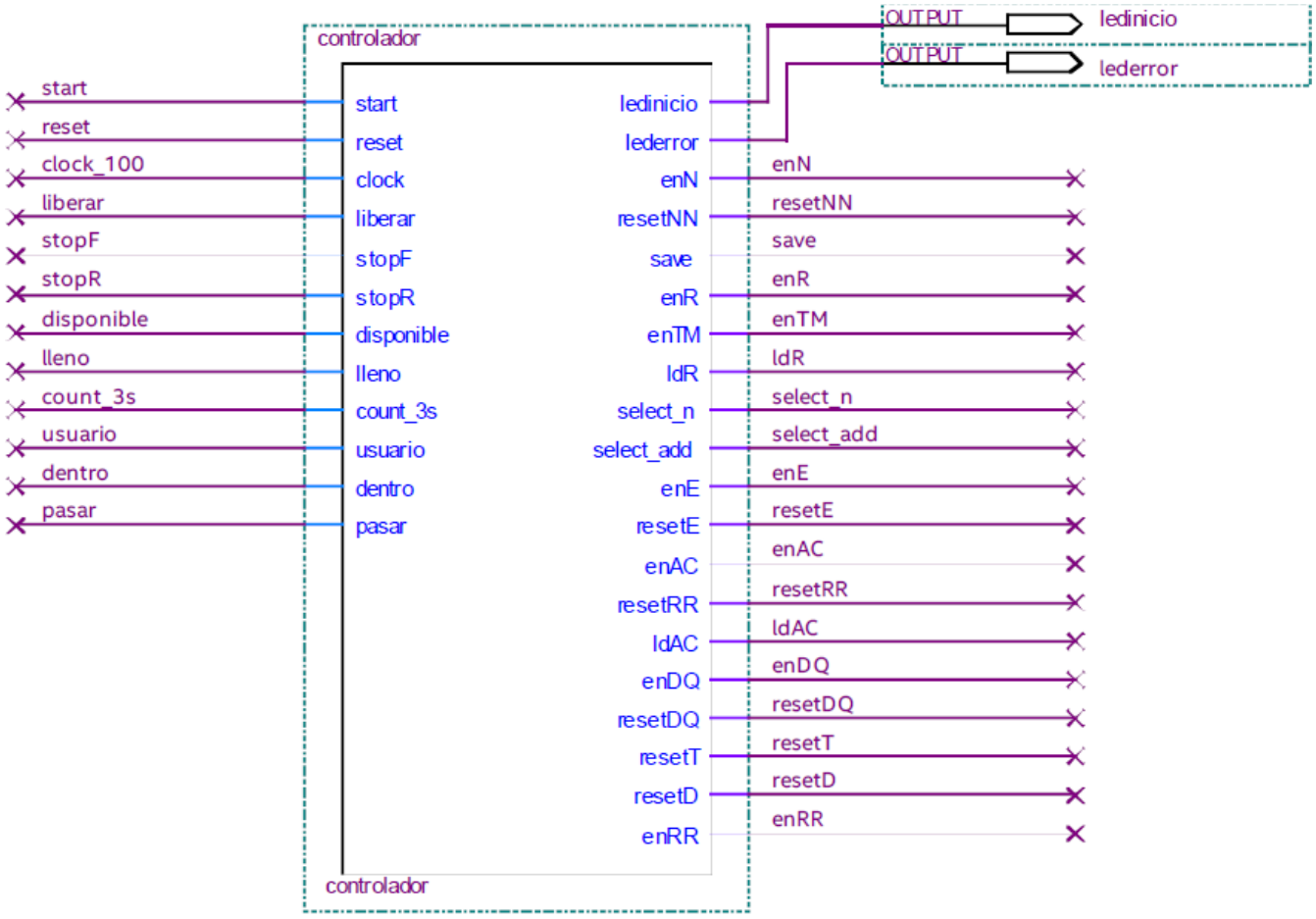


Centro de Atención a usuarios



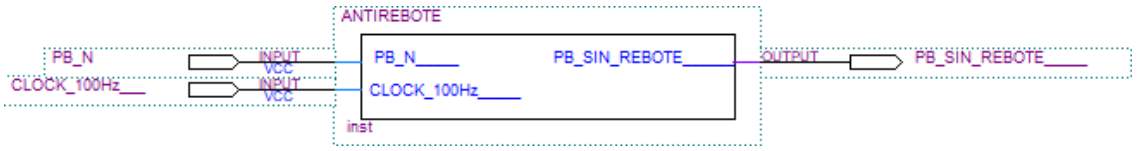
Centro de Atención a usuarios

Controlador



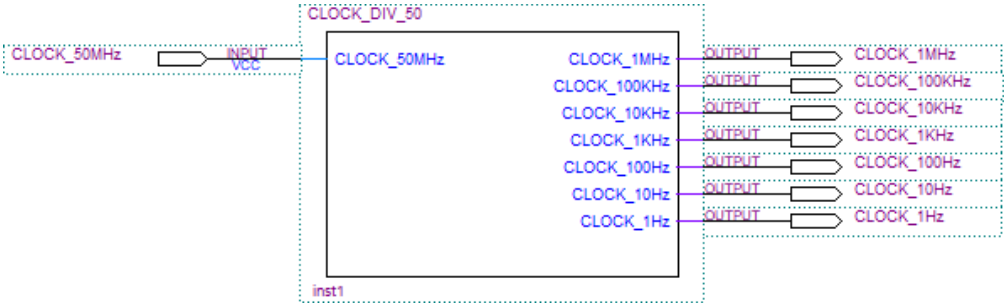
4) Listado que contenga todas las señales de entrada, salida y señales internas de cada bloque

Bloque Anti-rebote



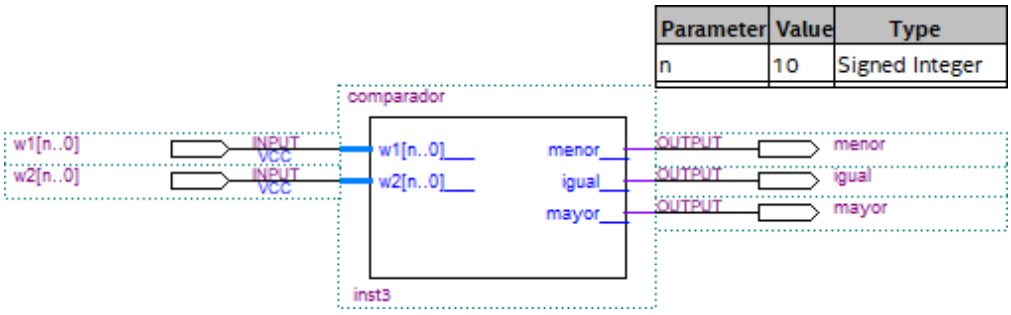
- PB_N: señal generada por la pulsación de un botón, la cual es limpiada para obtener pulsos ideales de dicha señal. En este caso, las señales que ingresan a este bloque son Start_n, reset_n, n_mods_n, usuario_n y pasar_n.
- CLOCK_100Hz: señal de reloj que permite el funcionamiento del bloque anti-rebote.
- PB_SIN_REBOTE: señal limpiada a partir de la entrada PB_N. Aquí se obtienen las señales Start, resetn, usuario, pasar, n_mods, usuario y pasar.

Bloque Clock_DIV_50



- CLOCK_50MHz: Señal de reloj de entrada cuya frecuencia será dividida. Esta señal debe tener una frecuencia de 50MHz.
- CLOCK_1MHz: Señal de reloj de salida con una frecuencia de 1MHz.
- CLOCK_100KHz: Señal de reloj de salida con una frecuencia de 100KHz.
- CLOCK_10KHz: Señal de reloj de salida con una frecuencia de 10KHz.
- CLOCK_1KHz: Señal de reloj de salida con una frecuencia de 1KHz, utilizada en las memorias RAM para su correcta lectura y escritura.
- CLOCK_100Hz: Señal de reloj de salida con una frecuencia de 100Hz, utilizada en todos los bloques sincrónicos.
- CLOCK_10Hz: Señal de reloj de salida con una frecuencia de 10Hz.
- CLOCK_1Hz: Señal de reloj de salida con una frecuencia de 1Hz, utilizada para contar segundos.

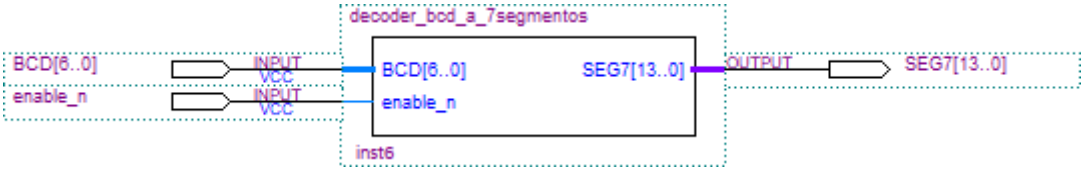
Bloque Comparador



Centro de Atención a usuarios

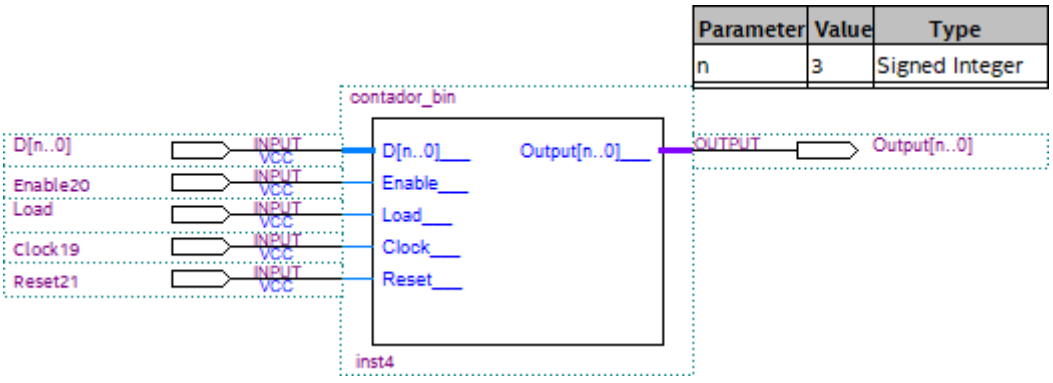
- W1: Primera señal a comparar.
- W2: Segunda señal a comparar.
- Menor: Señal de salida que indica que W1 es menor a W2.
- Igual: Señal de salida que indica que W1 es igual a W2. Esta salida se utiliza en el comparador cp_3seg para indicar que TMP es igual a 3, lo cual da a conocer que dicho contador de segundos ya contó 3 segundos.
- Mayor: Señal de salida que indica que W1 es mayor a W2. Esta salida se utiliza en los comparadores cp_lim_tur para obtener si n_client es mayor a 99, lo cual da a conocer que se han dado todos los 99 turnos a los clientes; cp_RAM_mods para detener el recorrido de la memoria RAM con la información de turnos asignados a los módulos y cp_ticket para obtener la señal dentro que indica que at_client es mayor a n_clientR para establecer que el cliente vaya al módulo con el ticket asignado correcto.

Bloque Decoder_bcd_a_7_segmentos



- BCD: Señal de entrada a convertir para ser mostrada en displays de 7 segmentos.
- Enable_n: Señal de lógica negativa para habilitar la conversión de la entrada en BCD.
- SEG7: Señales de salida a mostrar en displays.
- En este caso se convertirá la señal disp cuando endecon valga cero, obteniendo la salida dispQ a ser mostrada en 2 displays, al tener decena y unidades.

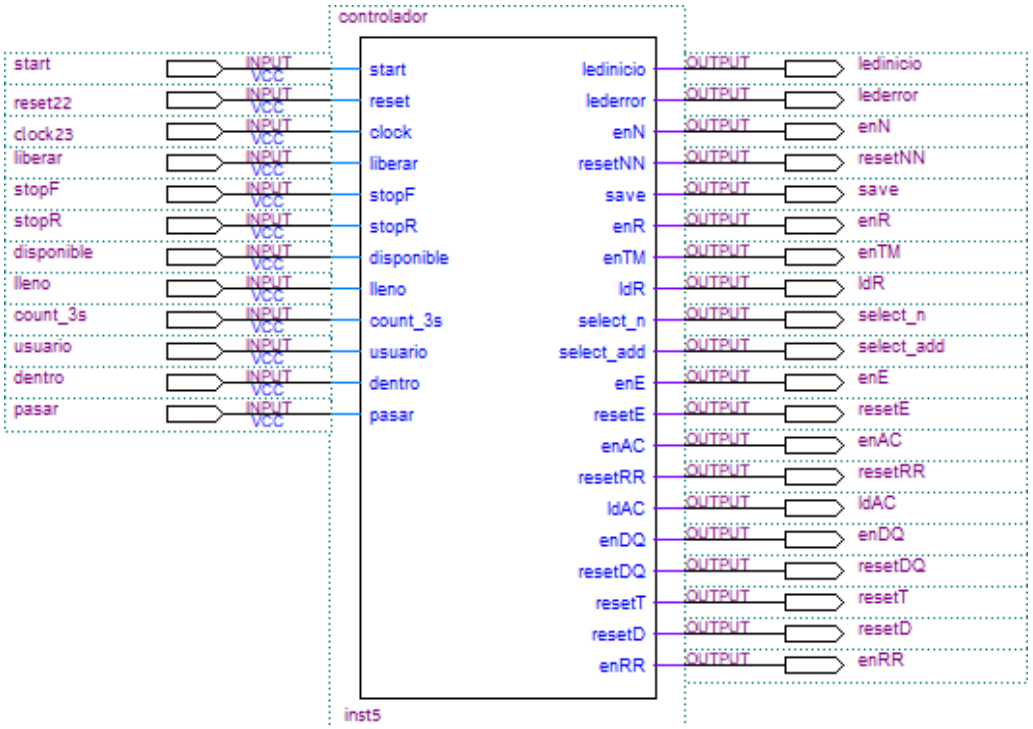
Bloque contador_bin



- D: Señal a ser cargada en paralelo en el contador up.
- Enable: Señal que permite habilitar el contador up.
- Load: Señal que permite hacer carga en paralelo al contador up, siempre cuando Enable y Load se habiliten a la vez.
- Clock: Señal de reloj que permite el funcionamiento del contador up.
- Reset: Señal que permite resetear el contador up.
- Output: Salida del contador up.
- Este bloque se utiliza en cnt_atendidos para contar los clientes atendidos, al cual se le hace carga en paralelo en 1 cuando enAC y ldAC estén habilitadas; en cnt_turnos para contar los turnos dados a los clientes cuando enN se habilite, reiniciándose con resetNN y cambiando la salida n_client; en cnt_it_mods para recorrer la memoria RAM, al cual se le hace carga en

paralelo en 1 cuando enR y ldR se habiliten a la vez, cambiando la salida addR; y en cnt_segL que permite contar segundos cuando enTM se habilite, reiniciándose con resetTM y cambiando la salida TMP.

Bloque controlador 1

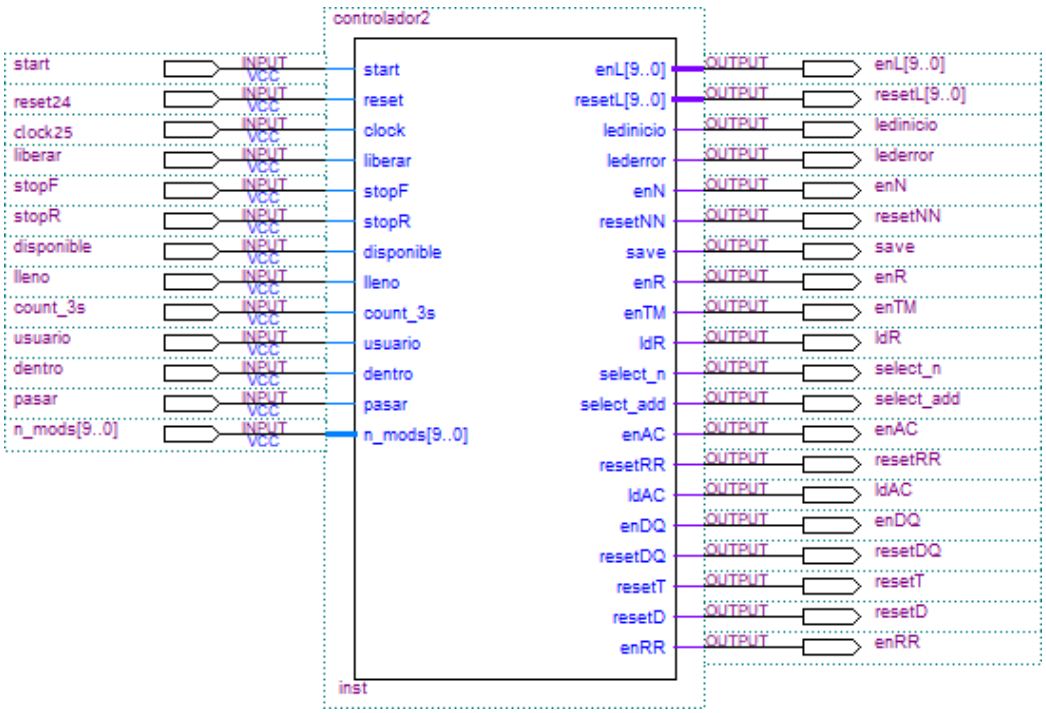


- start: permite resetear y habilitar los módulos de atención al usuario, estableciendo el sistema en un estado de activación donde se espera que un usuario pida su ticket con la señal usuario, que se libere algún módulo con la señal liberar o que pase un usuario a su respectivo módulo de atención con la señal pasar.
- resetn: Reinicia el sistema digital sin importar en qué estado se encuentre.
- clock: señal de reloj que permite los cambios de estados del sistema digital.
- liberar: botón que indica que un módulo desea liberar su puesto porque ya atendió a su respectivo cliente, asignándole un nuevo turno a atender siempre que sea posible. Se valida que el módulo de atención este ocupado en verdad y para la asignación de un nuevo turno al mismo que ya no se haya atendido a todos los 99 usuarios de límite.
- stopF: indica que se han atendido a 99 usuarios.
- stopR: detiene el recorrido de la memoria RAM que contiene el turno que le corresponde a cada módulo de atención.
- disponible: indica que un módulo se encuentra vacío.
- lleno: indica que todos los módulos están siendo ocupados por un usuario.
- count_3s: en caso que se genere un error cuenta tres segundos para encender un indicador led.
- usuario: permite darle un turno a un nuevo cliente solo si aún no se han dado 99 turnos y si existe un módulo vacío.
- dentro: se activa cuando el número del ticket es el mismo número que ha sido llamado por pantalla en el respectivo módulo de atención.
- pasar: indica cuando hay que atender al siguiente cliente; para cumplir con esto se debe verificar que el cliente se dirija al módulo correcto según el número de su ticket.
- ledinicio: señal que activa un diodo led indicando que el sistema se encuentra en el estado de activación, esperando por las señales usuario, liberar y pasar.

Centro de Atención a usuarios

- lederror: si no se encuentra el ticket asignado en ningún puesto, se da un error, de esta forma se valida que el número de turnos dados sea mayor a los atendidos.
- enN: permite habilitar el contador cnt_turno de turnos asignados a los clientes.
- resetNN: permite resetear el contador cnt_turno de turnos asignados a los clientes.
- resetTM: permite resetear el contador cnt_segL que cuenta 3 segundos para encender el led de error en caso de ser necesario.
- save: permite guardar un dato en la memoria RAM ram_mods con la información de los turnos asignados en cada módulo.
- enR: permite habilitar el contador cnt_it_mods que permite recorrer la RAM ram_mods.
- enTM: permite habilitar el contador cnt_segL.
- ldR: permite hacer carga en paralelo al contador cnt_it_mods.
- select_n: selectora del mux mux_cliente.
- select_add: selectora del mux mux_add.
- enE: permite habilitar el encendedor de leds leds_mods, de forma que se encienda el led según la dirección dada en dicho bloque.
- resetE: permite resetear el bloque encendedor de leds leds_mods.
- enAC: permite habilitar el contador cnt_atendidos con el número de tickets de los clientes que ya han sido atendidos.
- resetRR: permite reset el registro de sostenimiento reg_mods, que guarda el número del módulo que ya atendió su ticket.
- ldAC: permite hacer carga en paralelo al contador cnt_atendidos.
- enDQ: permite habilitar el registro de sostenimiento reg_modulo que guarda el módulo de atención que le corresponde al siguiente cliente.
- resetDQ: permite resetear el registro de sostenimiento reg_modulo.
- resetT: permite resetear el contador cnt_segD que cuenta 2 segundos de forma continua.
- resetD: permite resetear el registro de sostenimiento reg_selectD que permite el cambio de la selectora select_D del mux mux_disp para mostrar en los displays el número de turno y módulo que le corresponde al siguiente cliente de forma alternada.
- enRR: permite habilitar el registro de sostenimiento reg_mods que guarda el número del módulo que ya atendió su ticket.

Bloque controlador 2



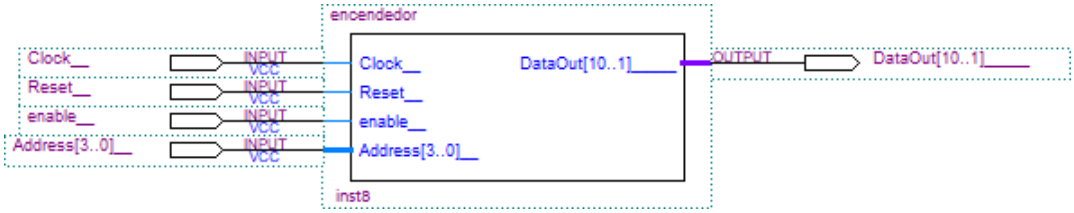
Centro de Atención a usuarios

- start: permite resetear y habilitar los módulos de atención al usuario, estableciendo el sistema en un estado de activación donde se espera que un usuario pida su ticket con la señal usuario, que se libere algún módulo con la señal liberar o que pase un usuario a su respectivo módulo de atención con la señal pasar.
- resetn: Reinicia el sistema digital sin importar en qué estado se encuentre.
- clock: señal de reloj que permite los cambios de estados del sistema digital.
- liberar: botón que indica que un módulo desea liberar su puesto porque ya atendió a su respectivo cliente, asignándole un nuevo turno a atender siempre que sea posible. Se valida que el módulo de atención este ocupado en verdad y para la asignación de un nuevo turno al mismo que ya no se haya atendido a todos los 99 usuarios de límite.
- stopF: indica que se han atendido a 99 usuarios.
- stopR: detiene el recorrido de la memoria RAM que contiene el turno que le corresponde a cada módulo de atención.
- disponible: indica que un módulo se encuentra vacío.
- lleno: indica que todos los módulos están siendo ocupados por un usuario.
- count_3s: en caso que se genere un error cuenta tres segundos para encender un indicador led.
- usuario: permite darle un turno a un nuevo cliente solo si aún no se han dado 99 turnos y si existe un módulo vacío.
- dentro: se activa cuando el número del ticket es el mismo número que ha sido llamado por pantalla en el respectivo módulo de atención.
- pasar: indica cuando hay que atender al siguiente cliente; para cumplir con esto se debe verificar que el cliente se dirija al módulo correcto según el número de su ticket.
- n_mods: Señal que representa un teclado de 10 botones, correspondiente cada uno a un módulo diferente y permite saber cuándo alguno requiere ser liberado.
- enL: Señal que permite habilitar 10 registros de sostenimiento que encienden 1 led cada uno, según el botón presionado del módulo de acuerdo a la señal de entrada n_mods.
- resetL: Señal que permite reiniciar los 10 registros de sostenimiento que representan los leds de los módulos según se requiera.
- ledinicio: señal que activa un diodo led indicando que el sistema se encuentra en el estado de activación, esperando por las señales usuario, liberar y pasar.
- lederror: si no se encuentra el ticket asignado en ningún puesto, se da un error, de esta forma se valida que el número de turnos dados sea mayor a los atendidos.
- enN: permite habilitar el contador cnt_turno de turnos asignados a los clientes.
- resetNN: permite resetear el contador cnt_turno de turnos asignados a los clientes.
- resetTM: permite resetear el contador cnt_segL que cuenta 3 segundos para encender el led de error en caso de ser necesario.
- save: permite guardar un dato en la memoria RAM ram_mods con la información de los turnos asignados en cada módulo.
- enR: permite habilitar el contador cnt_it_mods que permite recorrer la RAM ram_mods.
- enTM: permite habilitar el contador cnt_segL.
- ldR: permite hacer carga en paralelo al contador cnt_it_mods.
- select_n: selectora del mux mux_cliente.
- select_add: selectora del mux mux_add.
- enAC: permite habilitar el contador cnt_atendidos con el número de tickets de los clientes que ya han sido atendidos.
- resetRR: permite reset el registro de sostenimiento reg_mods, que guarda el número del módulo que ya atendió su ticket.
- ldAC: permite hacer carga en paralelo al contador cnt_atendidos.

Centro de Atención a usuarios

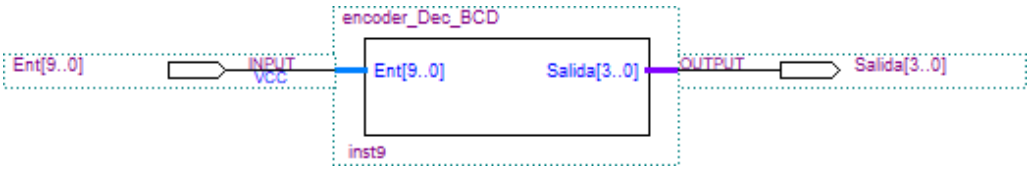
- enDQ: permite habilitar el registro de sostenimiento reg_modulo que guarda el módulo de atención que le corresponde al siguiente cliente.
- resetDQ: permite resetear el registro de sostenimiento reg_modulo.
- resetT: permite resetear el contador cnt_segD que cuenta 2 segundos de forma continua.
- resetD: permite resetear el registro de sostenimiento reg_selectD que permite el cambio de la selectora select_D del mux mux_disp para mostrar en los displays el número de turno y módulo que le corresponde al siguiente cliente de forma alternada.
- enRR: permite habilitar el registro de sostenimiento reg_mods que guarda el número del módulo que ya atendió su ticket.

Bloque encendedor



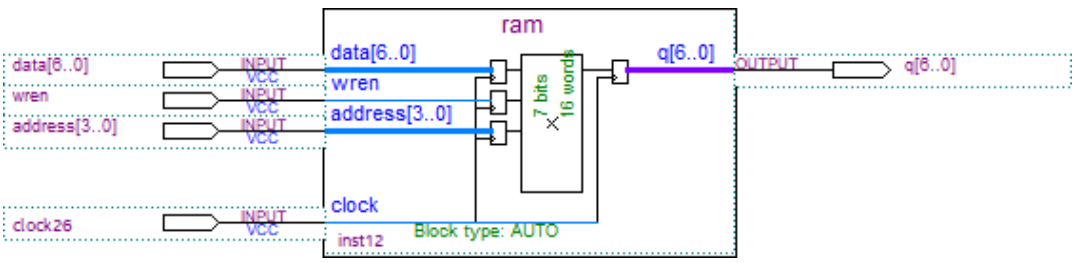
- clock: Señal de reloj que permite el funcionamiento del bloque.
- reset: Señal que permite reiniciar la salida DataOut, asignando todos sus bits en cero.
- enable: Señal que permite habilitar el bloque.
- address: Señal que indica la posición del bit a cambiar de la salida DataOut.
- DataOut: Señal de salida con bits establecidos en 1 o 0 según lo indicado en la señal address y si se ha habilitado la señal enable.
- Este bloque se utiliza en leds_mods, el cual de acuerdo a la señal addQ y cuando se habilite enE, cambiará un bit de la señal mods, la cual representa los 10 leds de cada módulo. Este bloque se reinicia con resetE.

Bloque encoder_Dec_BCD



- Ent: Señal a ser convertida de BCD a decimal en binario.
- Salida: Señal de salida ya convertida en binario.
- Este bloque se utiliza en enco_mods para saber el número del módulo que ya realizó la atención de su ticket asignado, convirtiendo la señal n_mods para obtener la salida addMQ.

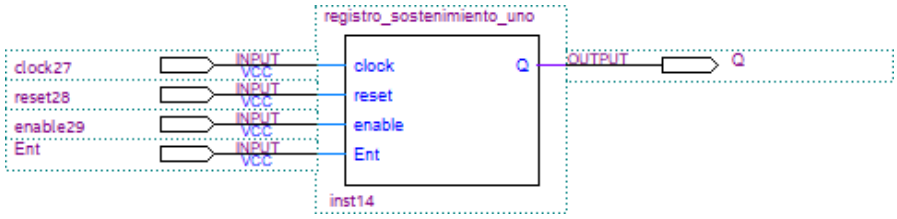
Bloque RAM



- Data: Señal que representa los datos a almacenar en la memoria RAM.

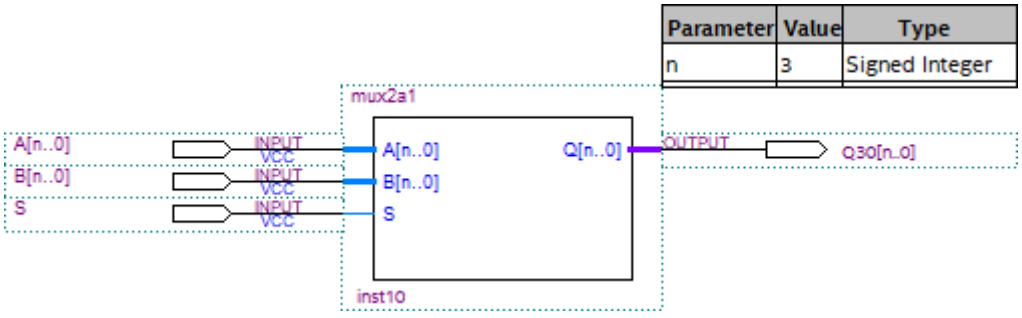
- Wren: Señal que permite habilitar la escritura en la memoria RAM.
- Address: Señal que permite determinar la posición de la memoria RAM a leer o escribir.
- Clock: Señal de reloj que permite el funcionamiento de la memoria RAM.
- Q: Señal de salida de la posición de la memoria RAM.
- Este bloque se utiliza en RAM_mods, la cual almacenará el número de ticket asignado a cada módulo, de acuerdo a la dirección addQ, obteniendo la salida n_clientR, permitiendo guardar los datos de n_clientQ cuando se habilite la señal save. La señal de reloj de este bloque debe tener una frecuencia mayor a la del bloque controlador para realizar escrituras y lecturas sin problemas, por eso se le asigna un clock de 1KHz.

Bloque registro_de_sostenemiento_uno



- Clock: Señal de reloj que permite el funcionamiento del bloque.
- Reset: Señal que permite reiniciar el bloque.
- Enable: Señal que permite habilitar el bloque.
- Ent: Señal de entrada a ser guardada en el bloque en la salida Q cuando Enable se habilite.
- Q: Señal de salida de la señal guardada en el bloque.
- Este bloque se utiliza en reg_módulo para guardar en la señal addDQ lo que haya en addQ cuando enDQ se habilite, reiniciándose con resetDQ, con esto se guarda el módulo de atención del siguiente turno a ser atendido; reg_mods para guardar en la señal addM lo que haya en addMQ cuando enRR se habilite, reiniciándose con resetRR, con esto se guarda el número de módulo que ya atendió su ticket; y en reg_selectD para cambiar el valor de la selectora select_D del mux mux_disp para mostrar de forma alternada el módulo y ticket de atención siguientes. Este bloque trabaja con señales de 1 bit.

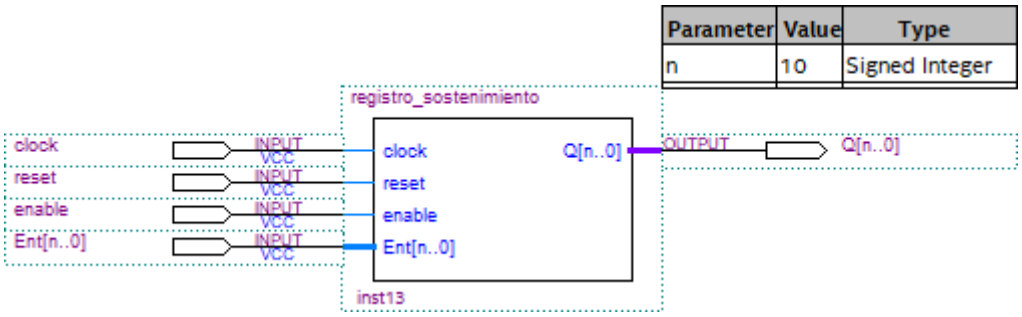
Bloque mux_2a1



- A: Señal A a ser multiplexada.
- B: Señal B a ser multiplexada.
- S: Selectora de señal de multiplexación.
- Q: Señal de salida.
- Este bloque se utiliza en mux_disp para intercambiar la información mostrada en los displays según la selectora select_D; en mux_add para alternar entre direcciones de las memorias RAM según la selectora select_add, permitiendo recorrer la RAM con la señal A addR o

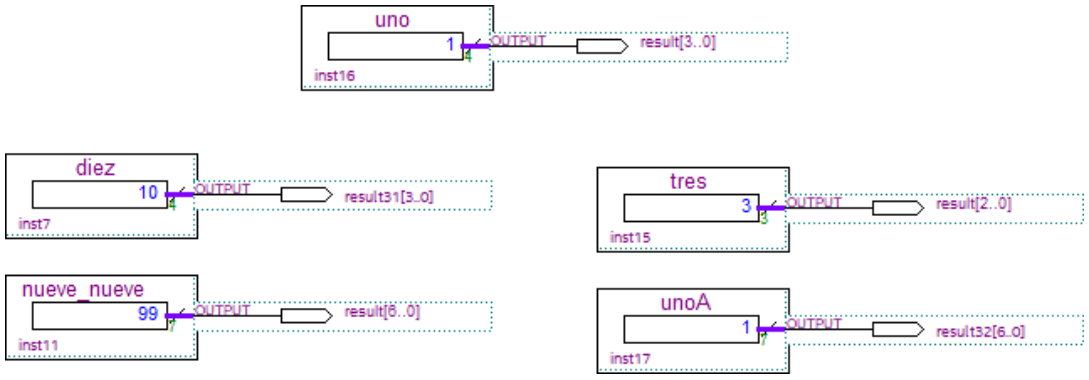
liberar un módulo con la señal B addM y en mux_cliente para asignar un número de ticket a un módulo según la memoria RAM y con la señal A n_client, o reiniciar el número de ticket de un módulo según la RAM con la señal B cuyo valor en cero.

Bloque registro_sostenimiento



- Clock: Señal de reloj que permite el funcionamiento del bloque.
- Reset: Señal que permite reiniciar el bloque.
- Enable: Señal que permite habilitar el bloque.
- Ent: Señal de entrada a ser guardada en el bloque en la salida Q cuando Enable se habilite.
- Q: Señal de salida de la señal guardada en el bloque.
- Este bloque se utiliza en reg_módulo para guardar en la señal addDQ lo que haya en addQ cuando enDQ se habilite, reiniciándose con resetDQ, con esto se guarda el módulo de atención del siguiente turno a ser atendido; reg_mods para guardar en la señal addM lo que haya en addMQ cuando enRR se habilite, reiniciándose con resetRR, con esto se guarda el número de módulo que ya atendió su ticket; y en reg_selectD para cambiar el valor de la selectora select_D del mux mux_disp para mostrar de forma alternada el módulo y ticket de atención siguientes. Este bloque trabaja con señales de más de 1 bit. Este bloque también se utiliza en la partición 2 en vez del bloque encendedor de leds para representar el encendido de los 10 leds de los 10 módulos de atención, habilitándose cada uno según la señal enL y reiniciándose con resetL.

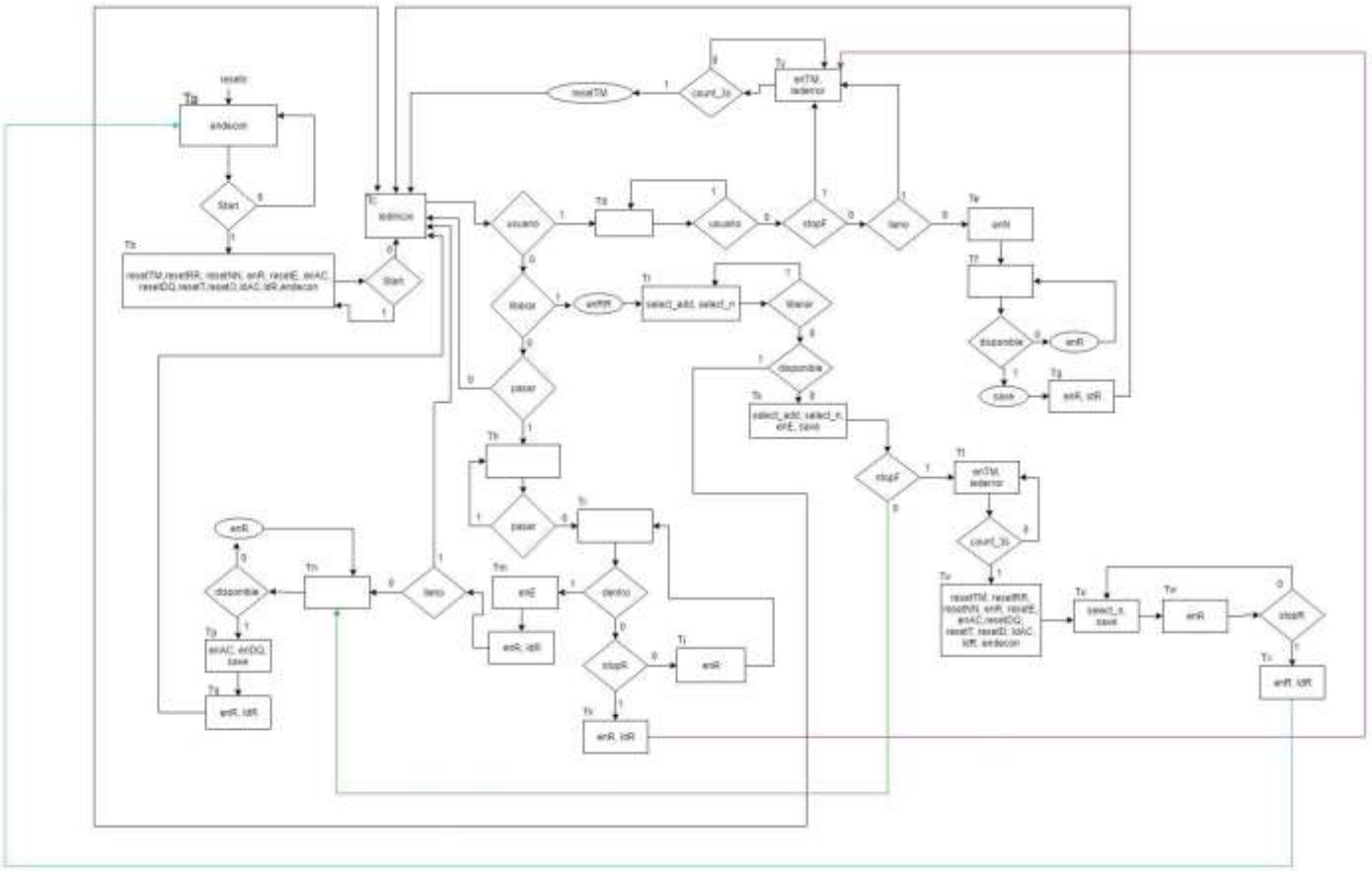
Bloques Numéricos



- Constantes numéricas en binario utilizadas para realizar comparaciones. Diez es utilizada para detener el recorrido de la RAM según el comparador cp_RAM_mods; uno y unoA se utilizan para hacer carga en paralelo a los contadores cnt_it_mods y cnt_atendidos; tres se utiliza en el comparador cp_3seg que determina que cnt_segL ha contado 3 segundos y

nueve_nueve se utiliza en el comparador cp_lim_tur para conocer si se ha dado todos los 99 turnos límites a los clientes.

5) Diagrama ASM del Controlador con su respectiva explicación a detalle



Indicando claramente todas las señales de entrada y salida utilizadas (con su respectiva explicación a detalle.

Estado	Explicación
Ta	Estado inicial del sistema. Se mantiene encendida la señal endecon para mantener los displays sin mostrar un número de ticket, es decir cero.
Tb	Presionando botón start, se reinician todos los bloques sincrónicos con sus respectivos reset o con carga en paralelo en caso de algunos contadores que inician su cuenta en 1.
Tc	Estado de activación del sistema. La señal ledinicio representa que el usuario puede elegir entre la opción usuario, liberar o pasar. En la opción usuario se da un turno a un cliente nuevo, en liberar se indica que un módulo de atención ya atendió a su cliente y puede desocuparse, y en pasar se realiza la atención del respectivo cliente que toque. Al activarse la señal liberar, se activa la salida enRR para guardar en un registro de sostenimiento el número del módulo que presionó su botón para liberarse.
Td	Presionando el botón de la señal usuario. Al soltarlo, se verifica que no se hay asignado los 99 turnos con la señal stopF y que exista un módulo vacio con la señal lleno, para la correcta asignación de un nuevo turno a un nuevo cliente.
Te	Habilitando la señal enN que habilita el contador de turnos para que aumente en uno su cuenta.

Tf	Aquí se recorre todas las posiciones de la memoria RAM que contiene la información del ticket asignado a cada módulo. Si esta vacío, se le asigna un ticket guardando esta información en la RAM con la señal save. Si esta vacío la señal disponible debe valer 1, caso contrario se va a la siguiente posición de la RAM activando el contador que permite iterar a la misma con la señal en R.
Tg	Reiniciando el contador que permite recorrer la memoria RAM, haciéndole carga en paralelo con enR y ldR.
Th	Estado para validar que se presione y suelte el botón pasar, en caso de mantenerlo presionado se mantiene en este estado, caso contrario se pasa al estado Ti.
Ti	Validando que el próximo ticket que toque atenderse se encuentre asignado en algún módulo, para esto se recorre la memoria RAM hasta que stopR valga 1, en caso de encontrar el ticket asignado en un módulo, dentro vale 1 y se pasa al estado Tm1. Si se terminó de recorrer toda la memoria RAM, lo que equivale a que stopR valga 1, se pasa al estado Tk.
Tj	Habilitando la señal enR para habiltar el contador que permite iterar la memoria RAM e ir a la siguiente dirección.
Tk	Habilitando enR y ldR para reiniciar el contador que permite iterar la memoria RAM haciendo carga en paralelo con el valor 1.
Tm1	Habilitando la señal enE, de forma que se encienda el diodo led correspondiente al módulo de atención que atenderá al cliente que toca y que ya lo tenía asignado.
Tm2	Habilitando enR y ldR para reiniciar el contador que permite iterar la memoria RAM haciendo carga en paralelo con el valor 1. Luego se pregunta por la señal lleno que para saber si hay un módulo vacío, si lo hay (lleno = 0), entonces se pasa al estado Tn para asignar un turno a ese módulo vacío, caso contrario se pasa al estado Tc.
Tn	Iterando la memoria RAM para buscar un módulo sin ticket asignado ya que ya pasó el cliente que le tocaba. Al encontrar ese módulo vacío, se pasa al estado Tp.
Tp	Habilitando el contador de turnos atendidos con la señal enAC, habilitando el registro de sostenimiento que muestra el número de módulo que le toca al siguiente turno con enDQ y asignando el turno al módulo vacío encontrado en Tn.
Tq	Habilitando enR y ldR para reiniciar el contador que permite iterar la memoria RAM haciendo carga en paralelo con el valor 1.
Tr	Habilitando las selectoras select_add y select_n para asignarle cero al ticket del módulo que desea liberarse y para acceder a la dirección del mismo en la RAM. Al desactivarse liberar, se pregunta si el módulo en verdad estaba vacío con disponible, si estaba vacio (disponible=1), entonces no es necesario liberar dicho módulo y se vuelve a lestado Tc, caso contrario se pasa al estado Ts para dicho procedimiento.
Ts	Habilitando las selectoras select_add, select_n y save para reiniciar el módulo, además de enE para apagar el diodo led del mismo. Al liberar el módulo deseado, se pregunta por la señal stopF para verificar si ya se atendieron los 99 turnos, en caso de que esto se haya cumplido (stopF=1) se pasa al estado Tt, caso contrario se pasa al estado Tn para volver a asignarle un turno al módulo liberado.
Tt	Habilitando un led de error con la señal lederror para indicar que ya se atendieron los 99 turnos de límite. Esto se hace por 3 segundos habilitando un contador de segundos con enTM. Al pasar los 3 segundos (count_3s=1) se pasa al estado Tv.

Tu	Habilitando la señal select_n y save para reiniciar cada posición de la memoria RAM de los módulos.
Tv	Habilitando todos los reset de todos los bloques sincrónicos, incluyendo contadores, registros de sostenimiento y el encendedor de leds.
Tw	Habilitando la señal enR para ir a la siguiente dirección de la memoria RAM. Luego se pregunta por stopR para saber si ya se terminó de recorrer toda la memoria RAM. En caso de que stopR=1 se pasa al estado Tx, caso contrario se pasa al estado Tu para reiniciar la siguiente dirección utilizada.
Tx	Habilitando enR y ldR para reiniciar el contador que permite iterar la memoria RAM haciendo carga en paralelo con el valor 1.
Ty	Estado parar encender un led de error con la señal ledError por 3 segundos. Los 3 segundos se cuentan con un contador con la señal enTM y se pregunta siempre si ya se contó estos 3 segundos con la señal count_3s. Si count_3s=1, se reinicia el contador de segundos con la señal resetTM y se pasa al estado Tc.

6) Código VHDL de la MSS del sistema digital

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY controlador IS
	PORT(start,reset,clock,liberar,stopF,stopR,disponible,
		lleno,count_3s,usuario,dentro,pasar,stopFQ,ult_cli,bit_ult,fin: IN STD_LOGIC;
		ledinicio,lederror,enN,resetNN,save,enR,enTM,ldR,
		select_n,select_add,enE,resetE,enAC,resetRR,ldAC,
		enDQ,resetDQ,resetT,resetD,resetTM,enRR,endecon,enF: OUT STD_LOGIC);
END controlador;

ARCHITECTURE sol OF controlador IS
Type estado is (Ta,Tb,Tc,Td,Te,Tf,Tg1,Tg2,Tg3,Th,Ti,Tj,Tk,Tm1,Tm2,Tn,Tp,Tq,
                                                Tr,Ts1,Ts2,Tt,Tu,Tv,Tw,Tx,Ty1,Ty2);
SIGNAL y:estado;
BEGIN
	PROCESS(clock,reset)
	BEGIN
		if reset='0' then y<=Ta;
		elsif (clock'event and clock='1') then
			case y is
				when Ta=>		if start='1' then y<=Tb;
                                                                else y<=Ta;
                                                                end if;
                                when Tb=>if start='1' then y<=Tb;
                                                                else y<=Tc;
                                                                end if;
                                when Tc=> if usuario='1' then y<=Td;
                                                                elsif liberar='1' then y<=Tr;
                                                                elsif pasar='1' then y<=Th; end if;
                                when Td=>if usuario='0' and stopF='0' and lleno='0' then y<=Te;
                                                                        elsif stopF='1' or lleno='1' then
y<=Ty1; end if;
                                when Te=>y<=Tf;
                                when Tf=> if disponible='1' then y<=Tg1; end if;
                                when Tg1=>		y<=Tg2;
                                when Tg2=>		y<=Tg3;
                                when Tg3=>		y<=Tc;
                                when Th=>if pasar='0' then y<=Ti; end if;
                                when Ti=> if dentro='0' and stopR='0' then y<=Tj;
                                                                        elsif dentro='1' and ult_cli='1' then
y<=Tq;
                                                                        elsif dentro='1' and ult_cli='0' then
y<=Tm1;
                                                                elsif stopR='1' then y<=Tk; end if;
                                when Tj=> y<=Ti;
                                when Tk=> y<=Ty1;
                                when Tm1=>		y<=Tm2;
                                when Tm2=>		if lleno='0' then y<=Tn;
                                                                else y<=Tc; end if;
                                when Tn=>if stopFQ='0' then y<=Tp;
                                                                else y<=Tc;end if;
                                when Tp=>y<=Tq;
                                when Tq=>y<=Tc;
                                when Tr=> if liberar='0' and bit_ult='0' then y<=Tc;
                                                                        elsif liberar='0' and bit_ult='1' then
y<=Ts1; end if;
                                when Ts1=>		y<=Ts2;
                                when Ts2=>		if fin='0' then y<=Tc;else
                                                                        y<=Tt; end if;
                                when Tt=> if count_3s='1' then y<=Tv; end if;
                                when Tu=>y<=Tw;
                                when Tv=>y<=Tu;
                                when Tw=>		if stopR='0' then y<=Tu;
                                                                else y<=Tx; end if;
                                when Tx=> y<=Ta;
                                when Ty1=>		if count_3s='1' then y<=Ty2; end if;
                                when Ty2=>		y<=Tc;
                        end case;
                end if;
        END PROCESS;
```

```
PROCESS(y,start,liberar,stopF,stopR,disponible,
                        lleno,count_3s,usuario,dentro,pasar)
        BEGIN
                        ledinicio<='0';lederror<='0';enN<='0';resetNN<='0';save<='0';
                        enR<='0';enTM<='0';ldR<='0';select_n<='0';select_add<='0';enE<='0';
                        resetE<='0';enAC<='0';resetRR<='0';ldAC<='0';enDQ<='0';resetDQ<='0';
                        resetT<='0';resetD<='0';resetTM<='0';enRR<='0';endecon<='0';enF<='0';
                        case y is
                                when Ta=>endecon<='1';
                                when Tb=>resetTM<='1';resetRR<='1';enN<='1';enR<='1';resetE<='1';

        enAC<='1';resetDQ<='1';resetT<='1';resetD<='1';ldAC<='1';
                                                                        ldR<='1';endecon<='1';
                                when Tc=> ledinicio<='1';
                                                                                if usuario='0' and liberar='1' then
enRR<='1'; end if;
                                when Td=>
                                when Te=>enN<='1';
                                when Tf=> if disponible='0' then enR<='1';end if;
                                when Tg1=>      save<='1';enF<='1';
                                when Tg2=>      enDQ<='1';
                                when Tg3=>      enR<='1';ldR<='1';
                                when Th=>
                                when Ti=>
                                when Tj=> enR<='1';
                                when Tk=> enR<='1';ldR<='1';
                                when Tm1=>      enE<='1';
                                when Tm2=>      enR<='1';ldR<='1';
                                when Tn=>
                                when Tp=>enAC<='1';
                                when Tq=>enR<='1';ldR<='1';enDQ<='1';
                                when Tr=> select_add<='1';select_n<='1';
                                when Ts1=>      select_add<='1';select_n<='1';enE<='1';save<='1';
                                when Ts2=>
                                when Tt=> enTM<='1';lederror<='1';
                                when Tu=>select_n<='1';save<='1';
                                when Tv=> resetTM<='1';resetRR<='1';resetNN<='1';enR<='1';

        resetE<='1';enAC<='1';resetDQ<='1';resetT<='1';

        resetD<='1';ldAC<='1';ldR<='1';endecon<='1';
                                when Tw=>       enR<='1';
                                when Tx=> enR<='1';ldR<='1';
                                when Ty1=>      enTM<='1';lederror<='1';
                                when Ty2=>      resetTM<='1';
                        end case;
        END PROCESS;
END sol;
```

- start: permite resetear y habilitar los módulos de atención al usuario, estableciendo el sistema en un estado de activación donde se espera que un usuario pida su ticket con la señal usuario, que se libere algún módulo con la señal liberar o que pase un usuario a su respectivo módulo de atención con la señal pasar.
- resetn: Reinicia el sistema digital sin importar en qué estado se encuentre.
- clock: señal de reloj que permite los cambios de estados del sistema digital.
- liberar: botón que indica que un módulo desea liberar su puesto porque ya atendió a su respectivo cliente, asignándole un nuevo turno a atender siempre que sea posible. Se valida que el módulo de atención este ocupado en verdad y para la asignación de un nuevo turno al mismo que ya no se haya atendido a todos los 99 usuarios de límite.
- stopF: indica que los 10 módulos de atención ya tienen asignado un número cliente para atender.
- stopR: detiene el recorrido de la memoria RAM que contiene el turno que le corresponde a cada módulo de atención.
- disponible: indica que un módulo se encuentra vacío.
- lleno: indica que todos los módulos están siendo ocupados por un usuario.
- count_3s: en caso que se genere un error cuenta tres segundos para encender un indicador led.

Centro de Atención a usuarios

- usuario: permite darle un turno a un nuevo cliente solo si aún no se han dado 99 turnos y si existe un módulo vacío.
- dentro: se activa cuando el número del ticket es el mismo número que ha sido llamado por pantalla en el respectivo módulo de atención.
- pasar: indica cuando hay que atender al siguiente cliente; para cumplir con esto se debe verificar que el cliente se dirija al módulo correcto según el número de su ticket.
- stopFQ: señal que indica que ya se asignaron los 99 clientes de límite.
- ult_cli: señal que indica que se encuentra atendiendo al cliente número 99 y se encuentra ocupado su módulo asignado, de forma que se valide que no se apague el led del módulo mencionado por error.
- bit_ult: señal que indica que el módulo que se desea liberar está ocupado o no.
- fin: señal que indica que ya se atendió a los 99 clientes de límite en el sistema.
- ledinicio: señal que activa un diodo led indicando que el sistema se encuentra en el estado de activación, esperando por las señales usuario, liberar y pasar.
- lederror: si no se encuentra el ticket asignado en ningún puesto, se da un error, de esta forma se valida que el número de turnos dados sea mayor a los atendidos.
- enN: permite habilitar el contador cnt_turno de turnos asignados a los clientes.
- resetNN: permite resetear el contador cnt_turno de turnos asignados a los clientes.
- resetTM: permite resetear el contador cnt_segL que cuenta 3 segundos para encender el led de error en caso de ser necesario.
- save: permite guardar un dato en la memoria RAM ram_mods con la información de los turnos asignados en cada módulo.
- enR: permite habilitar el contador cnt_it_mods que permite recorrer la RAM ram_mods.
- enTM: permite habilitar el contador cnt_segL.
- ldR: permite hacer carga en paralelo al contador cnt_it_mods.
- select_n: selectora del mux mux_cliente.
- select_add: selectora del mux mux_add.
- enE: permite habilitar el encendedor de leds leds_mods, de forma que se encienda el led según la dirección dada en dicho bloque.
- resetE: permite resetear el bloque encendedor de leds leds_mods.
- enAC: permite habilitar el contador cnt_atendidos con el número de tickets de los clientes que ya han sido atendidos.
- resetRR: permite reset el registro de sostenimiento reg_mods, que guarda el número del módulo que ya atendió su ticket.
- ldAC: permite hacer carga en paralelo al contador cnt_atendidos.
- enDQ: permite habilitar el registro de sostenimiento reg_modulo que guarda el módulo de atención que le corresponde al siguiente cliente.
- resetDQ: permite resetear el registro de sostenimiento reg_modulo.
- resetT: permite resetear el contador cnt_segD que cuenta 2 segundos de forma continua.
- resetD: permite resetear el registro de sostenimiento reg_selectD que permite el cambio de la selectora select_D del mux mux_disp para mostrar en los displays el número de turno y módulo que le corresponde al siguiente cliente de forma alternada.
- enRR: permite habilitar el registro de sostenimiento reg_mods que guarda el número del módulo que ya atendió su ticket.
- enF: permite al bloque encendedor guardar el número de ticket asignado a un módulo de atención y asignar su respectivo bit que indica que ya tiene un cupo asignado al mismo módulo.

7) Código vhdl de todos los bloques usados

VHDL REGISTRO_SOSTENIMIENTO

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
USE  IEEE.Std_logic_arith.all;

ENTITY registro_sostenimiento IS
	generic (n:INTEGER:=10);
	PORT(clock,reset,enable: IN STD_LOGIC;
		Ent : IN STD_LOGIC_VECTOR(n downto 0);
		Q : OUT STD_LOGIC_VECTOR (n downto 0));
END registro_sostenimiento;

ARCHITECTURE sol OF registro_sostenimiento IS
SIGNAL temp: STD_LOGIC_VECTOR(n downto 0);
BEGIN
	PROCESS(clock,reset)
	BEGIN
		if reset='1' then temp<=(others => '0');
		elsif (clock'event and clock='1') then
			if(enable='1') then
				temp<=Ent;
			end if;
		end if;
	end process;
	Q<=temp;
END sol;
```

- Clock: Señal de reloj que permite el funcionamiento del bloque.
- Reset: Señal que permite reiniciar el bloque.
- Enable: Señal que permite habilitar el bloque.
- Ent: Señal de entrada a ser guardada en el bloque en la salida Q cuando Enable se habilite.
- Q: Señal de salida de la señal guardada en el bloque.
- Este bloque se utiliza en reg_módulo para guardar en la señal addDQ lo que haya en addQ cuando enDQ se habilite, reiniciándose con resetDQ, con esto se guarda el módulo de atención del siguiente turno a ser atendido; reg_mods para guardar en la señal addM lo que haya en addMQ cuando enRR se habilite, reiniciándose con resetRR, con esto se guarda el número de módulo que ya atendió su ticket; y en reg_selectD para cambiar el valor de la selectora select_D del mux mux_disp para mostrar de forma alternada el módulo y ticket de atención siguientes. Este bloque trabaja con señales de más de 1 bit. Este bloque también se utiliza en la partición 2 en vez del bloque encendedor de leds para representar el encendido de los 10 leds de los 10 módulos de atención, habilitándose cada uno según la señal enL y reiniciándose con resetL.

Centro de Atención a usuarios

VHDL MUX2a1

```
library ieee;
use ieee.std_logic_1164.all;

Entity mux2a1 is
generic (n:INTEGER:=3);
PORT( A: IN std_logic_vector(n downto 0);
                B: IN std_logic_vector(n downto 0);
                S: IN std_logic;
                Q: OUT std_logic_vector(n downto 0));
end mux2a1;

Architecture sol of mux2a1 is
Begin
        with S select
        Q<=A when '0',
                B when '1',
                A when others;
end sol;
```

- A: Señal A a ser multiplexada.
- B: Señal B a ser multiplexada.
- S: Selectora de señal de multiplexación.
- Q: Señal de salida.
- Este bloque se utiliza en mux_disp para intercambiar la información mostrada en los displays según la selectora select_D; en mux_add para alternar entre direcciones de las memorias RAM según la selectora select_add, permitiendo recorrer la RAM con la señal A addR o liberar un módulo con la señal B addM y en mux_cliente para asignar un número de ticket a un módulo según la memoria RAM y con la señal A n_client, o reiniciar el número de ticket de un módulo según la RAM con la señal B cuyo valor en cero.

VHDL COMPARADOR

```
library ieee;
use ieee.std_logic_1164.all;

Entity comparador is
        generic (n:INTEGER:=10);
        Port( w1: in std_logic_vector(n downto 0);
                        w2: in std_logic_vector(n downto 0);
                        menor,igual, mayor: out std_logic);
end comparador;

Architecture sol of comparador is
Begin
        mayor<='1' when w1>w2 else '0';
        menor<='1' when w1<w2 else '0';
        igual<='1' when w1=w2 else '0';
end sol;
```

W1: Primera señal a comparar.

W2: Segunda señal a comparar.

Menor: Señal de salida que indica que W1 es menor a W2.

Igual: Señal de salida que indica que W1 es igual a W2. Esta salida se utiliza en el comparador cp_3seg para indicar que TMP es igual a 3, lo cual da a conocer que dicho contador de segundos ya contó 3 segundos.

Mayor: Señal de salida que indica que W1 es mayor a W2. Esta salida se utiliza en los comparadores cp_lim_tur para obtener si n_client es mayor a 99, lo cual da a conocer que se han dado todos los 99 turnos a los clientes; cp_RAM_mods para detener el recorrido de la memoria RAM con la información de turnos asignados a los módulos y cp_ticket para obtener la señal dentro que indica que at_client es mayor a n_clientR para establecer que el cliente vaya al módulo con el ticket asignado correcto.

VHDL ENCODER_DEC_BCD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity encoder_Dec_BCD is
	Port (Ent: in STD_LOGIC_VECTOR(9 downto 0);
		Salida : OUT STD_LOGIC_VECTOR(3 downto 0));
end encoder_Dec_BCD;

Architecture sol of encoder_Dec_BCD is
signal Temp:STD_LOGIC_VECTOR(3 downto 0);
Begin
	process(Ent)
	Begin
                                        --0987654321
		if Ent="0000000001" then Temp<="0001";
		elsif Ent="0000000010" then Temp<="0010";
		elsif Ent="0000000100" then Temp<="0011";
		elsif Ent="0000001000" then Temp<="0100";
		elsif Ent="0000010000" then Temp<="0101";
		elsif Ent="0000100000" then Temp<="0110";
		elsif Ent="0001000000" then Temp<="0111";
		elsif Ent="0010000000" then Temp<="1000";
		elsif Ent="0100000000" then Temp<="1001";
		elsif Ent="1000000000" then Temp<="0000";
		else Temp<="0000"; end if;
	end process;
	Salida<=Temp;
end sol;
```

- Ent: Señal a ser convertida de BCD a decimal en binario.
- Salida: Señal de salida ya convertida en binario.
- Este bloque se utiliza en enco_mods para saber el número del módulo que ya realizó la atención de su ticket asignado, convirtiendo la señal n_mods para obtener la salida addMQ.

VHDL ENCENDEDOR

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

entity encendedor is
	Port (Clock: in std_logic;
		Reset		: in  STD_LOGIC:='0';
		enableA	:   in   STD_LOGIC:='0';
		enableB : in  STD_LOGIC:='0';
		Address	: in STD_LOGIC_VECTOR (3 downto 0):="0000";
		num		: in STD_LOGIC_VECTOR (6 downto 0):="0000000";
		pos_mod : out STD_LOGIC_VECTOR (3 downto 0):="0000";
		DataOut	: out STD_LOGIC_VECTOR (10 downto 1);
		posOut	: out STD_LOGIC_VECTOR (10 downto 1);
		ultimo : out STD_LOGIC;
		fin		: out std_logic);
end encendedor;

architecture Behavioral of encendedor is
signal temp: std_logic_vector(10 downto 1):="0000000000";
signal temp_pos: std_logic_vector(10 downto 1):="0000000000";
signal pos_1: std_logic_vector(6 downto 0):="0000000";
signal pos_2: std_logic_vector(6 downto 0):="0000000";
signal pos_3: std_logic_vector(6 downto 0):="0000000";
signal pos_4: std_logic_vector(6 downto 0):="0000000";
signal pos_5: std_logic_vector(6 downto 0):="0000000";
signal pos_6: std_logic_vector(6 downto 0):="0000000";
signal pos_7: std_logic_vector(6 downto 0):="0000000";
signal pos_8: std_logic_vector(6 downto 0):="0000000";
signal pos_9: std_logic_vector(6 downto 0):="0000000";
signal pos_10: std_logic_vector(6 downto 0):="0000000";
signal turnos: std_logic_vector(6 downto 0):="0000001";
begin
	process (Clock, Reset)
	begin
		if Reset='1' then
			temp<="0000000000";
			temp_pos<="0000000000";
			turnos<="0000001";
		elsif(Clock'event and Clock='1') then
			if (enableB='1' and Address>"0000") then
				temp_pos(conv_integer(Address))<=not(temp_pos(conv_integer(Address)));
				if Address="0001" then pos_1<=num;
				elsif Address="0010" then pos_2<=num;
				elsif Address="0011" then pos_3<=num;
				elsif Address="0100" then pos_4<=num;
				elsif Address="0101" then pos_5<=num;
				elsif Address="0110" then pos_6<=num;
				elsif Address="0111" then pos_7<=num;
				elsif Address="1000" then pos_8<=num;
				elsif Address="1001" then pos_9<=num;
				elsif Address="1010" then pos_10<=num; end if;end if;
			if (enableA='1' and Address>"0000") then
				if temp(conv_integer(Address))='0' then turnos<=turnos+"0000001"; end if;
				if temp(conv_integer(Address))='1' then
					temp_pos(conv_integer(Address))<=not(temp_pos(conv_integer(Address)));
					if Address="0001" then pos_1<="0000000";
				elsif Address="0010" then pos_2<="0000000";
				elsif Address="0011" then pos_3<="0000000";
				elsif Address="0100" then pos_4<="0000000";
				elsif Address="0101" then pos_5<="0000000";
				elsif Address="0110" then pos_6<="0000000";
				elsif Address="0111" then pos_7<="0000000";
				elsif Address="1000" then pos_8<="0000000";
				elsif Address="1001" then pos_9<="0000000";
				elsif Address="1010" then pos_10<="0000000"; end if;
				end if;
				temp(conv_integer(Address))<=not(temp(conv_integer(Address)));
			end if;
		end if;
	end process;
	pos_mod<="0001" when turnos=pos_1 else
					"0010" when turnos=pos_2 else
					"0011" when turnos=pos_3 else
					"0100" when turnos=pos_4 else
					"0101" when turnos=pos_5 else
					"0110" when turnos=pos_6 else
					"0111" when turnos=pos_7 else
					"1000" when turnos=pos_8 else
					"1001" when turnos=pos_9 else
					"1010" when turnos=pos_10 else "0000";
	fin<='1' when turnos>="1100011" else '0';
	DataOut<=temp;
	posOut<=temp_pos;
	ultimo<=temp(conv_integer(Address));
end Behavioral;
```

- clock: Señal de reloj que permite el funcionamiento del bloque.
- reset: Señal que permite reiniciar la salida DataOut, asignando todos sus bits en cero.
- enable: Señal que permite habilitar el bloque.

- address: Señal que indica la posición del bit a cambiar de la salida DataOut.
- DataOut: Señal de salida con bits establecidos en 1 o 0 según lo indicado en la señal address y si se ha habilitado la señal enable.
- Este bloque se utiliza en leds_mods, el cual de acuerdo a la señal addQ y cuando se habilite enE, cambiará un bit de la señal mods, la cual representa los 10 leds de cada módulo. Este bloque se reinicia con resetE.

VHDL DECODER_BCD A 7 SEGMENTOS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity decoder_bcd_a_7segmentos is
	Port ( BCD: in STD_LOGIC_VECTOR(6 downto 0);
		enable_n: std_logic;
		SEG7: OUT STD_LOGIC_VECTOR(13 downto 0));
end decoder_bcd_a_7segmentos;

Architecture sol of decoder_bcd_a_7segmentos is
signal mod1:STD_LOGIC_VECTOR(3 downto 0);
signal numA:STD_LOGIC_VECTOR(3 downto 0);
signal numB:STD_LOGIC_VECTOR(3 downto 0);
Begin
	process(BCD,mod1,numA,numB)
	begin
		mod1<=conv_std_logic_vector(conv_integer(BCD) MOD 10,4);
		numA<=conv_std_logic_vector(conv_integer(BCD) / 10,4);
		numB<=mod1;
	end process;
			--abcdefg
		SEG7(13 downto 7) <=	"0000001"	when numA = "0000" and enable_n='0' ELSE --0
					"1001111"	when numA = "0001" and enable_n='0' ELSE --1
					"0010010"	when numA = "0010" and enable_n='0' ELSE --2
					"0000110"	when numA = "0011" and enable_n='0' ELSE --3
					"1001100"	when numA = "0100" and enable_n='0' ELSE --4
					"0100100"	when numA = "0101" and enable_n='0' ELSE --5
					"0100000"	when numA = "0110" and enable_n='0' ELSE --6
					"0001111"	when numA = "0111" and enable_n='0' ELSE --7
					"0000000"	when numA = "1000" and enable_n='0' ELSE --8
					"0000100"	when numA = "1001" and enable_n='0' ELSE --9
					"0000001";
		SEG7(6 downto 0) <=	"0000001"	when numB = "0000" and enable_n='0' ELSE --0
					"1001111"	when numB = "0001" and enable_n='0' ELSE --1
					"0010010"	when numB = "0010" and enable_n='0' ELSE --2
					"0000110"	when numB = "0011" and enable_n='0' ELSE --3
					"1001100"	when numB = "0100" and enable_n='0' ELSE --4
					"0100100"	when numB = "0101" and enable_n='0' ELSE --5
					"0100000"	when numB = "0110" and enable_n='0' ELSE --6
					"0001111"	when numB = "0111" and enable_n='0' ELSE --7
					"0000000"	when numB = "1000" and enable_n='0' ELSE --8
					"0000100"	when numB = "1001" and enable_n='0' ELSE --9
					"0000001";
end sol;
```

- BCD: Señal de entrada a convertir para ser mostrada en displays de 7 segmentos.
- Enable_n: Señal de lógica negativa para habilitar la conversión de la entrada en BCD.
- SEG7: Señales de salida a mostrar en displays.
- En este caso se convertirá la señal disp cuando endecon valga cero, obteniendo la salida dispQ a ser mostrada en 2 displays, al tener decena y unidades.

Centro de Atención a usuarios

VHDL CONTADOR BINARIO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador_bin is
        generic (n:INTEGER:=3);
  port( D: in std_logic_vector(n downto 0);
                        Enable: in std_logic;
                        Load: in std_logic;
                        Clock: in std_logic;
                        Reset: in std_logic;
                        Output: out std_logic_vector(n downto 0));
end contador_bin;

architecture sol of contador_bin is
  signal temp: std_logic_vector(n downto 0);
begin
process(Clock,Reset)
  begin
    if Reset='1' then
      temp <= (others => '0');
            elsif(Clock'event and Clock='1') then
                        if Enable='1' then
                                    if Load='1' then
                                                temp <= D;
                                    elsif temp = conv_std_logic_vector((2**(n+1))-1,n+1) then
                                                temp <= (others => '0');
                                    else
                                                temp <= temp + 1;
                                    end if;
                        end if;
            end if;
  end process;
        output<=temp;
 end sol;
```

- D: Señal a ser cargada en paralelo en el contador up.
- Enable: Señal que permite habilitar el contador up.
- Load: Señal que permite hacer carga en paralelo al contador up, siempre cuando Enable y Load se habiliten a la vez.
- Clock: Señal de reloj que permite el funcionamiento del contador up.
- Reset: Señal que permite resetear el contador up.
- Output: Salida del contador up.
- Este bloque se utiliza en cnt_atendidos para contar los clientes atendidos, al cual se le hace carga en paralelo en 1 cuando enAC y ldAC estén habilitadas; en cnt_turnos para contar los turnos dados a los clientes cuando enN se habilite, reiniciándose con resetNN y cambiando la salida n_client; en cnt_it_mods para recorrer la memoria RAM, al cual se le hace carga en paralelo en 1 cuando enR y ldR se habiliten a la vez, cambiando la salida addR; y en cnt_segL que permite contar segundos cuando enTM se habilite, reiniciándose con resetTM y cambiando la salida TMP.

VHDL CLOCK_DIV_50

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY CLOCK_DIV_50 IS
   PORT
   (  CLOCK_50MHz    :IN STD_LOGIC;
      CLOCK_1MHz     :OUT STD_LOGIC;
      CLOCK_100KHz :OUT STD_LOGIC;
      CLOCK_10KHz    :OUT STD_LOGIC;
      CLOCK_1KHz     :OUT STD_LOGIC;
      CLOCK_100Hz    :OUT STD_LOGIC;
      CLOCK_10Hz     :OUT STD_LOGIC;
      CLOCK_1Hz      :OUT  STD_LOGIC);
END CLOCK_DIV_50;

ARCHITECTURE a OF CLOCK_DIV_50 IS
   SIGNAL count_1Mhz: STD_LOGIC_VECTOR(5 DOWNTO 0);
   SIGNAL count_100Khz, count_10Khz, count_1Khz: STD_LOGIC_VECTOR(2 DOWNTO 0);
   SIGNAL count_100hz, count_10hz, count_1hz: STD_LOGIC_VECTOR(2 DOWNTO 0);
   SIGNAL clock_1Mhz_int, clock_100Khz_int, clock_10Khz_int, clock_1Khz_int: STD_LOGIC;
   SIGNAL clock_100hz_int, clock_10hz_int, clock_1hz_int: STD_LOGIC;

BEGIN
   PROCESS
   BEGIN
-- Divide by 50
      WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1'; -- 24 Mhz
        IF count_1Mhz < 49 THEN
          count_1Mhz <= count_1Mhz + 1;
        ELSE
          count_1Mhz <= "000000";
        END IF;
        IF count_1Mhz < 4 THEN
          clock_1Mhz_int <= '0';
        ELSE
          clock_1Mhz_int <= '1';
        END IF;
      -- Ripple clocks are used in this code to save prescalar hardware
      -- Sync all clock prescalar outputs back to master clock signal
        clock_1Mhz <= clock_1Mhz_int;
        clock_100Khz <= clock_100Khz_int;
        clock_10Khz <= clock_10Khz_int;
        clock_1Khz <= clock_1Khz_int;
        clock_100hz <= clock_100hz_int;
        clock_10hz <= clock_10hz_int;
        clock_1hz <= clock_1hz_int;
      END PROCESS;
```

```
-- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
        IF count_100Khz /= 4 THEN
          count_100Khz <= count_100Khz + 1;
        ELSE
          count_100Khz <= "000";
          clock_100Khz_int <= NOT clock_100Khz_int;
        END IF;
    END PROCESS;
      -- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
        IF count_10Khz /= 4 THEN
          count_10Khz <= count_10Khz + 1;
        ELSE
          count_10Khz <= "000";
          clock_10Khz_int <= NOT clock_10Khz_int;
        END IF;
    END PROCESS;
      -- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
        IF count_1Khz /= 4 THEN
          count_1Khz <= count_1Khz + 1;
        ELSE
          count_1Khz <= "000";
          clock_1Khz_int <= NOT clock_1Khz_int;
        END IF;
    END PROCESS;
      -- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
        IF count_100hz /= 4 THEN
          count_100hz <= count_100hz + 1;
        ELSE
          count_100hz <= "000";
          clock_100hz_int <= NOT clock_100hz_int;
        END IF;
    END PROCESS;
      -- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
        IF count_10hz /= 4 THEN
          count_10hz <= count_10hz + 1;
        ELSE
          count_10hz <= "000";
          clock_10hz_int <= NOT clock_10hz_int;
        END IF;
    END PROCESS;
      -- Divide by 10
    PROCESS
    BEGIN
      WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
        IF count_1hz /= 4 THEN
          count_1hz <= count_1hz + 1;
        ELSE
          count_1hz <= "000";
          clock_1hz_int <= NOT clock_1hz_int;
        END IF;
    END PROCESS;
END a;
```

- CLOCK_50MHz: Señal de reloj de entrada cuya frecuencia será dividida. Esta señal debe tener una frecuencia de 50MHz.
- CLOCK_1MHz: Señal de reloj de salida con una frecuencia de 1MHz.
- CLOCK_100KHz: Señal de reloj de salida con una frecuencia de 100KHz.
- CLOCK_10KHz: Señal de reloj de salida con una frecuencia de 10KHz.

- CLOCK_1KHz: Señal de reloj de salida con una frecuencia de 1KHz, utilizada en las memorias RAM para su correcta lectura y escritura.
- CLOCK_100Hz: Señal de reloj de salida con una frecuencia de 100Hz, utilizada en todos los bloques sincrónicos.
- CLOCK_10Hz: Señal de reloj de salida con una frecuencia de 10Hz.
- CLOCK_1Hz: Señal de reloj de salida con una frecuencia de 1Hz, utilizada para contar segundos.

VHDL ANTIREBOTE

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

	-- Debounce Pushbutton: Filters out mechanical switch bounce for around 40Ms.
ENTITY ANTIREBOTE IS
	PORT(PB_N, CLOCK_100Hz		: IN	STD_LOGIC;
		PB_SIN_REBOTE		: OUT	STD_LOGIC);
END ANTIREBOTE;

ARCHITECTURE a OF ANTIREBOTE IS
	SIGNAL SHIFT_PB			: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

	-- Debounce clock should be approximately 10ms or 100Hz
	PROCESS
	BEGIN
		WAIT UNTIL (clock_100Hz'EVENT) AND (clock_100Hz = '1');
		-- Use a shift register to filter switch contact bounce
		SHIFT_PB(2 DOWNTO 0) <= SHIFT_PB(3 DOWNTO 1);
		SHIFT_PB(3) <= NOT PB_N;
		IF SHIFT_PB(3 DOWNTO 0)="0000" THEN
			PB_SIN_REBOTE <= '0';
		ELSE
			PB_SIN_REBOTE <= '1';
		END IF;
	END PROCESS;
END a;
```

- PB_N: señal generada por la pulsación de un botón, la cual es limpiada para obtener pulsos ideales de dicha señal. En este caso, las señales que ingresan a este bloque son Start_n, reset_n, n_mods_n, usuario_n y pasar_n.
- CLOCK_100Hz: señal de reloj que permite el funcionamiento del bloque anti-rebote.
- PB_SIN_REBOTE: señal limpiada a partir de la entrada PB_N. Aquí se obtienen las señales Start, resetn, usuario, pasar, n_mods, usuario y pasar.

VHDL REGISTRO DE SOSTENIMIENTO UNO

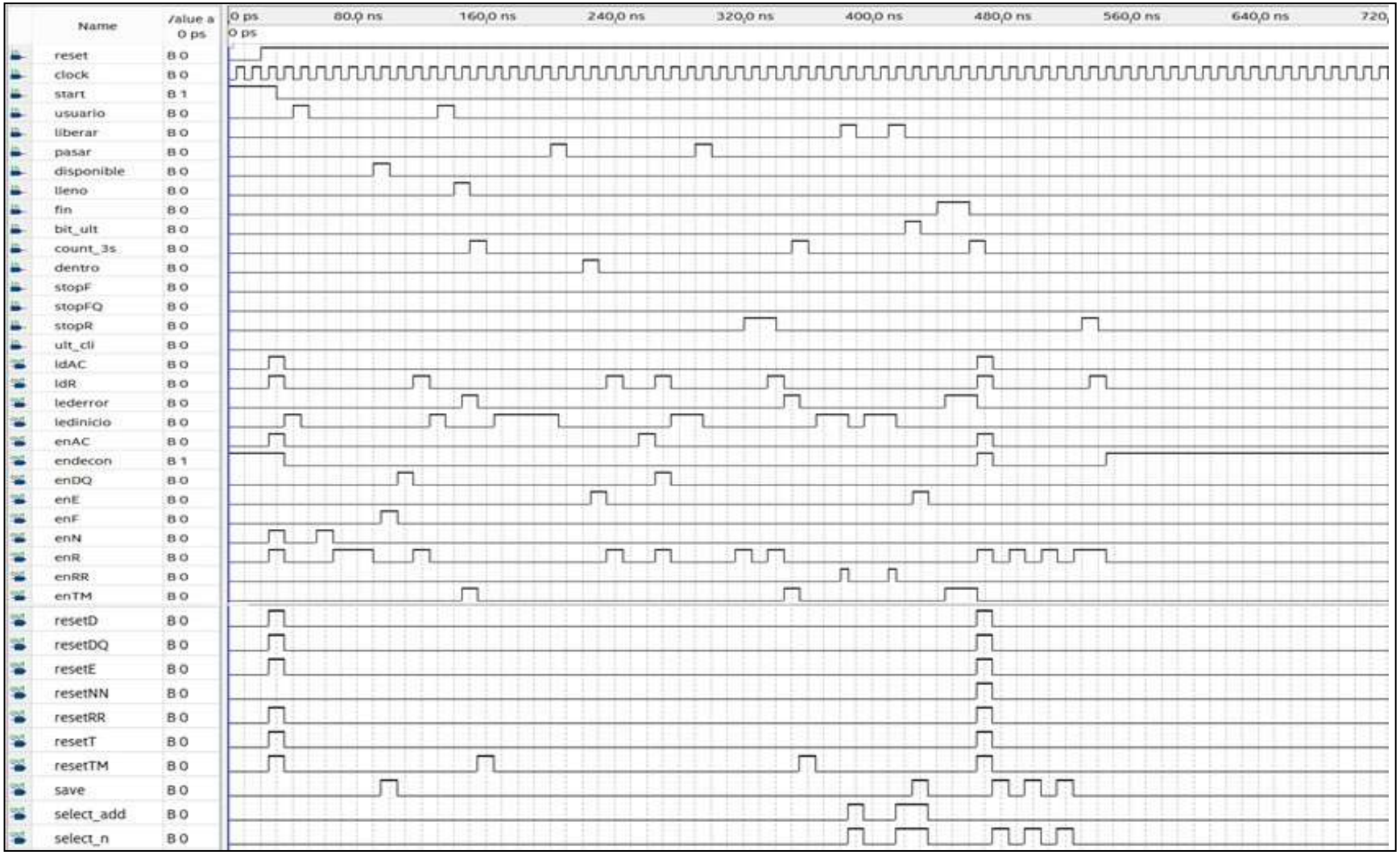
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
USE IEEE.Std_logic_arith.all;

ENTITY registro_sostenimiento_uno IS
        PORT(clock,reset,enable: IN STD_LOGIC;
                Ent : IN STD_LOGIC;
                Q : OUT STD_LOGIC);
END registro_sostenimiento_uno;

ARCHITECTURE sol OF registro_sostenimiento_uno IS
SIGNAL temp: STD_LOGIC;
BEGIN
        PROCESS(clock,reset)
        BEGIN
                if reset='1' then temp<= '0';
                elsif (clock'event and clock='1') then
                        if(enable='1') then
                                temp<=Ent;
                        end if;
                end if;
        end process;
        Q<=temp;
END sol;
```

- Clock: Señal de reloj que permite el funcionamiento del bloque.
- Reset: Señal que permite reiniciar el bloque.
- Enable: Señal que permite habilitar el bloque.
- Ent: Señal de entrada a ser guardada en el bloque en la salida Q cuando Enable se habilite.
- Q: Señal de salida de la señal guardada en el bloque.
- Este bloque se utiliza en reg_módulo para guardar en la señal addDQ lo que haya en addQ cuando enDQ se habilite, reiniciándose con resetDQ, con esto se guarda el módulo de atención del siguiente turno a ser atendido; reg_mods para guardar en la señal addM lo que haya en addMQ cuando enRR se habilite, reiniciándose con resetRR, con esto se guarda el número de módulo que ya atendió su ticket; y en reg_selectD para cambiar el valor de la selectora select_D del mux mux_disp para mostrar de forma alternada el módulo y ticket de atención siguientes. Este bloque trabaja con señales de 1 bit.

8) Diagrama de tiempo del controlador

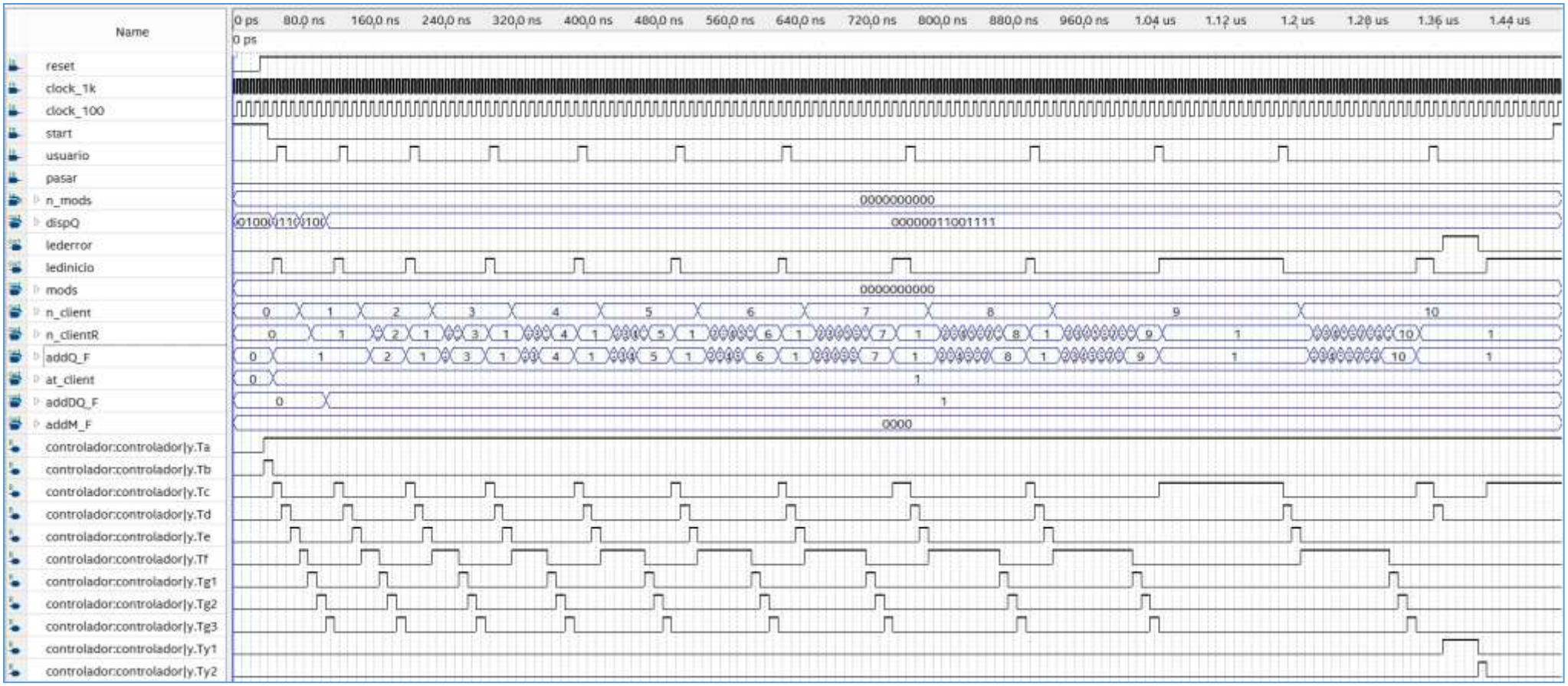


En el diagrama de tiempo mostrado, inicialmente reset tiene un valor de 0, por lo que, aunque está habilitada la señal Start el sistema no se inicializa cambiando de estado, una vez reset cambia su valor a 1 el sistema cambia al estado Tb al presionar Start y luego al estado Tc al soltar Start, habilitándose la salida ledinicio. Luego se presiona el botón usuario para dar un turno, si se habilita la entrada disponible. Luego se presiona el botón usuario de nuevo, pero ahora se habilita la entrada lleno, que indica que todos los puestos de atención están ocupados y no es posible dar un turno, por lo que se enciende el lederror por 3 segundos. Luego se presiona el botón pasar para dar paso al siguiente cliente, siempre que se habilite la entrada dentro. Pero si se presiona el botón pasar y se termina de buscar en todos los puestos de atención al turno que toque, habilitándose la entrada stopR, se pasa al estado Tk y luego al estado Ty para encender el lederror por 3 segundos. Finalmente, al habilitarse la entrada liberar y bit_ult, se pasa al estado Ts1 para reiniciar el módulo de atención deseado, y en caso de que fin tenga un valor de 1, se procede a reiniciar el sistema y se vuelve al estado Ta, habilitando la salida endecon.

9) Diagrama de tiempo del sistema digital.

Probando límite de expeder solo 10 cupos

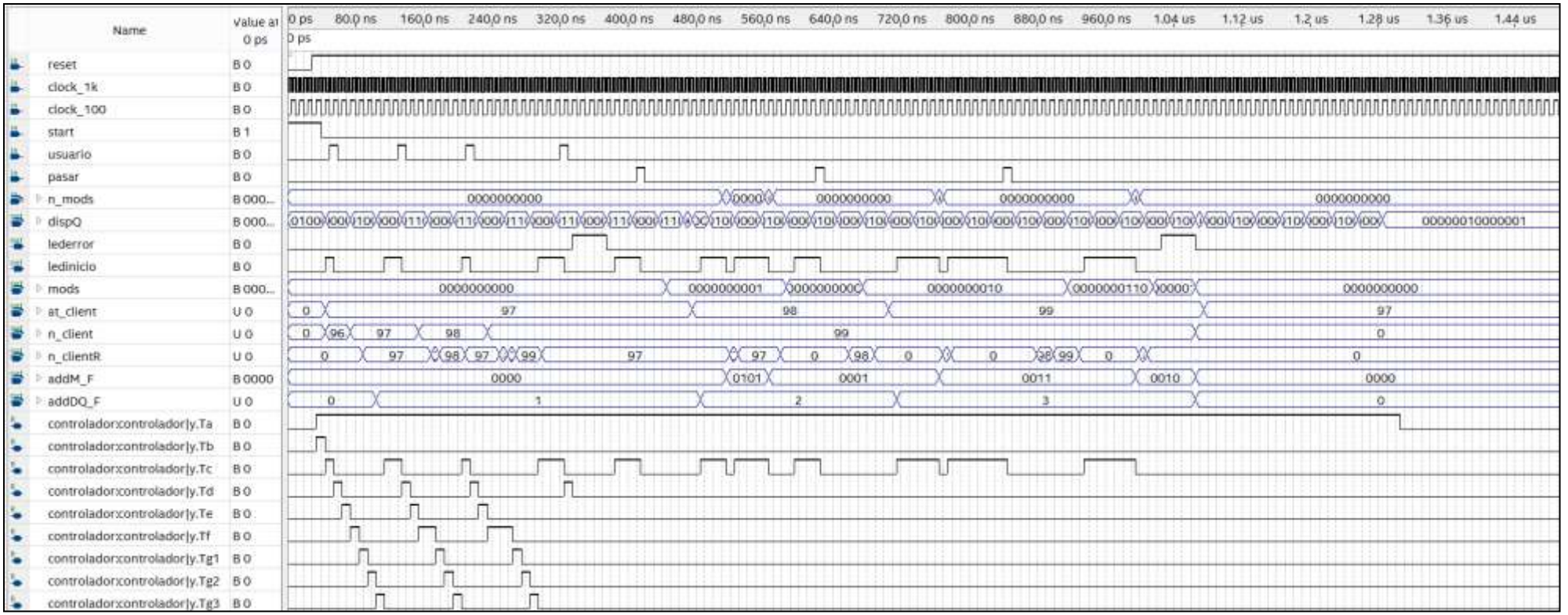
Centro de Atención a usuarios



En el diagrama de tiempo mostrado, luego de presionar y soltar Start con reset con un valor de 1, se procede a presionar y soltar el botón usuario 10 veces para expender 10 turnos, para lo cual los 10 módulos de atención ya tendrán asignado un turno y al querer expender otro turno, se encenderá el lederror por 3 segundos ya que no se puede asignar un turno más a un cliente si todos los módulos de atención tienen un turno ya asignado. Para verificar los valores guardados en la memoria RAM, se tiene la salida n_clientR con su dirección dada por la señal addQ_F, la cual empieza en cero al estar en la dirección cero de la RAM. Luego cuando se asigna el turno 1, se lo realiza al módulo 1 representado en la dirección 1 de la RAM, por lo que en dicha posición se guarda el turno 1; luego cuando se asigna el turno 2, se lo realiza al módulo 2 representado en la dirección 2 de la RAM, por lo que en dicha posición se guarda el turno 2, y así sucesivamente se realiza la asignación de los turnos del 1 al 10 en la memoria RAM.

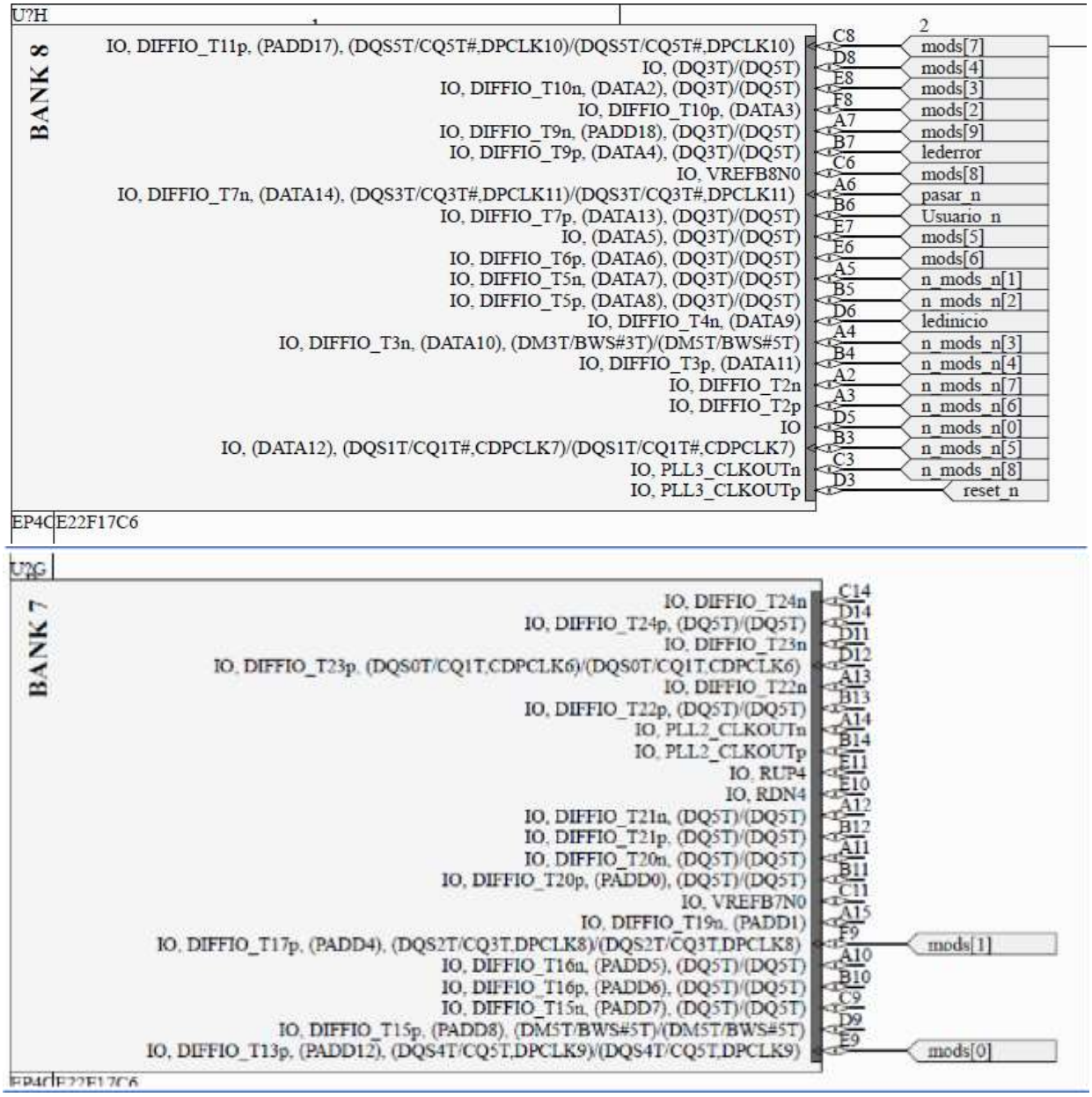
Probando hasta 99 clientes

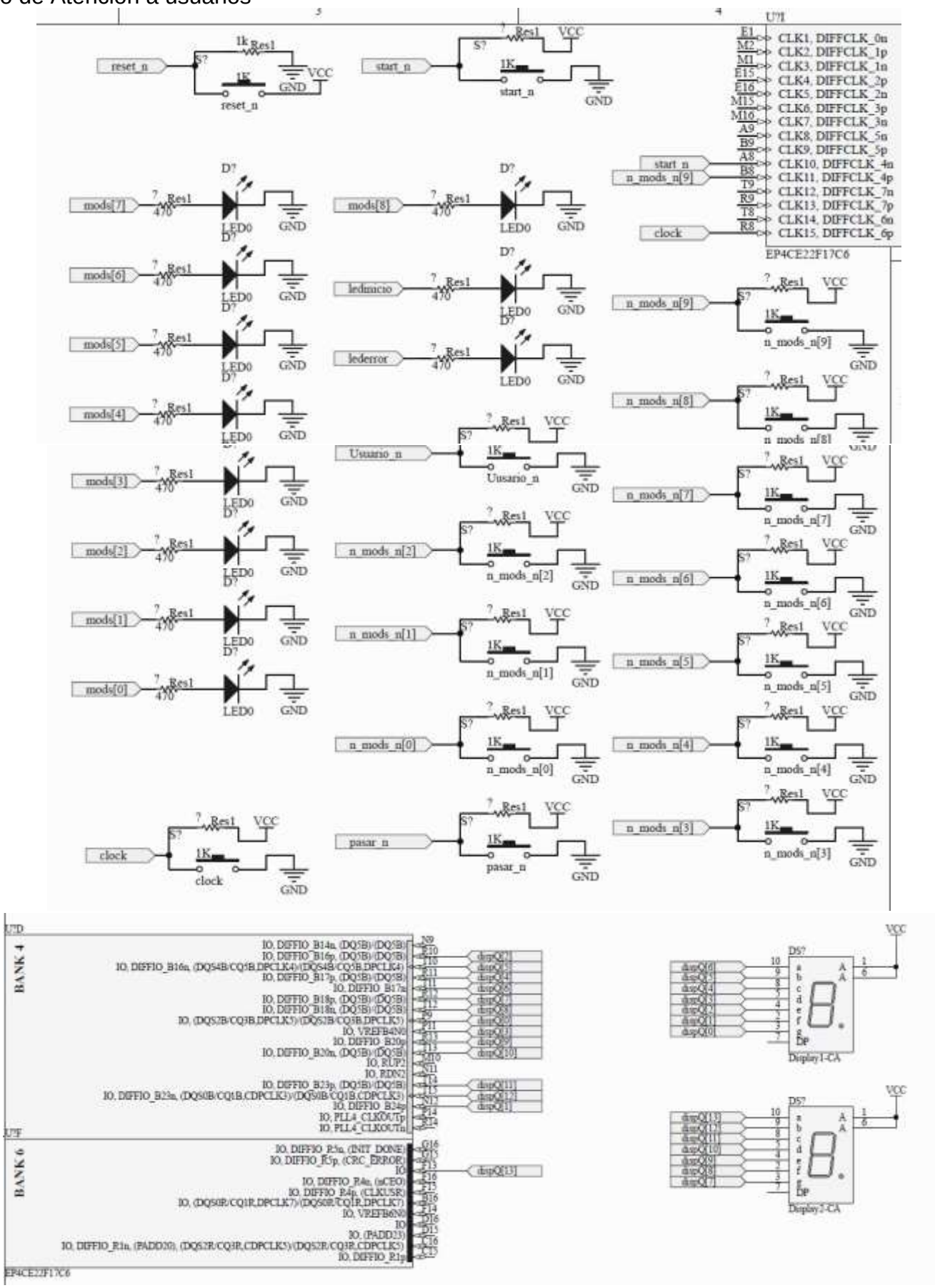
Centro de Atención a usuarios



En el diagrama de tiempo mostrado se expenden turnos a partir del 97, luego se procede a tender a estos 3 usuarios, y cada módulo se encarga de presionar su botón para liberarse de estar ocupado. Luego de haber atendido al cliente 99 y el módulo que lo haya atendido se haya liberado, en este caso es el módulo 3, se debe reiniciar el sistema para volver al estado inicial ya que ya se han atendido los 99 clientes de límite del sistema. Para verificar los valores guardados en la memoria RAM, se tiene la salida n_clientR con su dirección dada por la señal addQ_F, la cual empieza en cero al estar en la dirección cero de la RAM. Luego cuando se asigna el turno 97, se lo realiza al módulo 1 representado en la dirección 1 de la RAM, por lo que en dicha posición se guarda el turno 1; luego cuando se asigna el turno 98, se lo realiza al módulo 2 representado en la dirección 2 de la RAM, por lo que en dicha posición se guarda el turno 2, y así sucesivamente se realiza la asignación de los turnos del 97 al 99 en la memoria RAM.

10) Diagrama Esquemático del proyecto en físico elaborado con la herramienta ALTIUM Designer.




U?I
CLK1, DIFFCLK_0n
CLK2, DIFFCLK_1p
CLK3, DIFFCLK_1n
CLK4, DIFFCLK_2p
CLK5, DIFFCLK_2n
CLK6, DIFFCLK_3p
CLK7, DIFFCLK_3n
CLK8, DIFFCLK_5n
CLK9, DIFFCLK_5p
CLK10, DIFFCLK_4n
CLK11, DIFFCLK_4p
CLK12, DIFFCLK_7n
CLK13, DIFFCLK_7p
CLK14, DIFFCLK_6n
CLK15, DIFFCLK_6p
EP4CE22F17C6
reset_n
start_n
clock
mods[7]
mods[6]
mods[5]
mods[4]
mods[3]
mods[2]
mods[1]
mods[0]
ledmucio
lederror
Uusario_n
pasar_n
n_mods_n[9]
n_mods_n[8]
n_mods_n[7]
n_mods_n[6]
n_mods_n[5]
n_mods_n[4]
n_mods_n[3]
n_mods_n[2]
n_mods_n[1]
n_mods_n[0]
Res1
LED0
GND
VCC
U?D
U?F
BANK 4
BANK 6
EP4CE22F17C6
dispQ[0]
dispQ[1]
dispQ[2]
dispQ[3]
dispQ[4]
dispQ[5]
dispQ[6]
dispQ[7]
dispQ[8]
dispQ[9]
dispQ[10]
dispQ[11]
dispQ[12]
dispQ[13]
D?
Display1-CA
Display2-CA

CONCLUSIONES

- Se alcanzó de manera eficiente el control de atención a los usuarios, utilizando los bloques respectivos para el desarrollo correcto del mismo.
- Al realizar el presente proyecto se logró diseñar el control de un centro de usuarios compuesto de manera correcta por 10 módulos de atención, realizando la respectiva simulación se pudo observar que el diagrama de tiempo realizaba la ejecución correcta al iniciar con la botonera start y posteriormente realizando la asignación de módulos y dejando libres los mismos.
- Durante la ejecución del código, se plantearon diferentes bloques encargados de la administración y ejecución de errores y asignación de clientes a los debidos módulos, se creó un bloque encendedor el cual se tiene como funcionalidad ser un registro el cuál habilite cada led de cada módulo respectivo según se asigne cada usuario.

11) Anexos

https://github.com/ArianaOchoa/PROYECTODIGITALES2.git