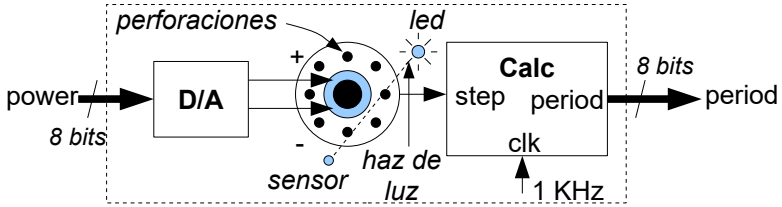
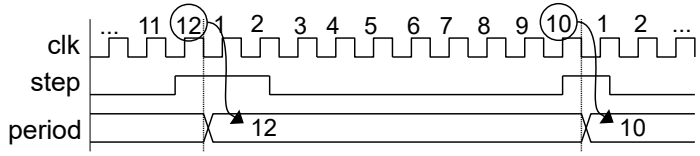
Pregunta 1



El dispositivo de la figura es un motor con control de velocidad digital. Recibe como entrada un valor numérico (*power* de 8 bits) que indica la potencia que se debe aplicar al motor. Un convertidor digital/analógico convierte este valor a una corriente que alimenta el motor. El motor hace girar un volante con perforaciones que dejan pasar la luz de un led hacia un sensor. Este sensor genera una señal digital que es la entrada (*step*) de un circuito **Calc** que calcula el periodo del motor como el número de ciclos de un reloj de 1 KHz que transcurrieron entre el paso de la luz por las dos últimas perforaciones. Este período es la salida del dispositivo (*period*).

Utilizando diseño modular de circuitos, implemente el módulo **Calc**. Considere que una perforación deja pasar la luz (*step*=1) durante al menos un ciclo del reloj, pero que a velocidades lentas, pueden transcurrir 2 o más ciclos como lo muestra el diagrama de tiempo de más abajo. Si requiere un circuito secuencial, basta con especificar su diagrama de estados.



Ayuda: Necesita tomar acción cuando *step* fue 0 en el ciclo previo y ahora es 1. Por lo tanto use un registro para almacenar el valor de *step* en el ciclo previo.

Pregunta 2

Escriba en system verilog una versión sintetizable del mismo módulo **Calc** de la pregunta 1.

Pregunta 3

I. Diseñe libremente una interfaz de entrada/salida para LRV32IM de manera que un programa pueda (a) establecer la entrada *power* que se aplica al convertidor digital/analógico de la pregunta 1 (use un registro de 8 bits), (b) leer la potencia aplicada, y (c) leer el valor de *period*. Ud. define las direcciones de los puertos de entrada/salida que va a requerir. Su interfaz debe hacer uso del bus de LRV32IM: *rd*, *wr*, *clk*, *be3-be0*, *a31-a0* y *d31-d0*.

II. Programe la función: *void setPeriod(unsigned char period)*

Esta función debe usar la interfaz de la parte *I* para probar con distintos niveles de potencia hasta lograr, en la medida de lo posible, el periodo *period*. La función retorna cuando se alcanzó el periodo deseado.

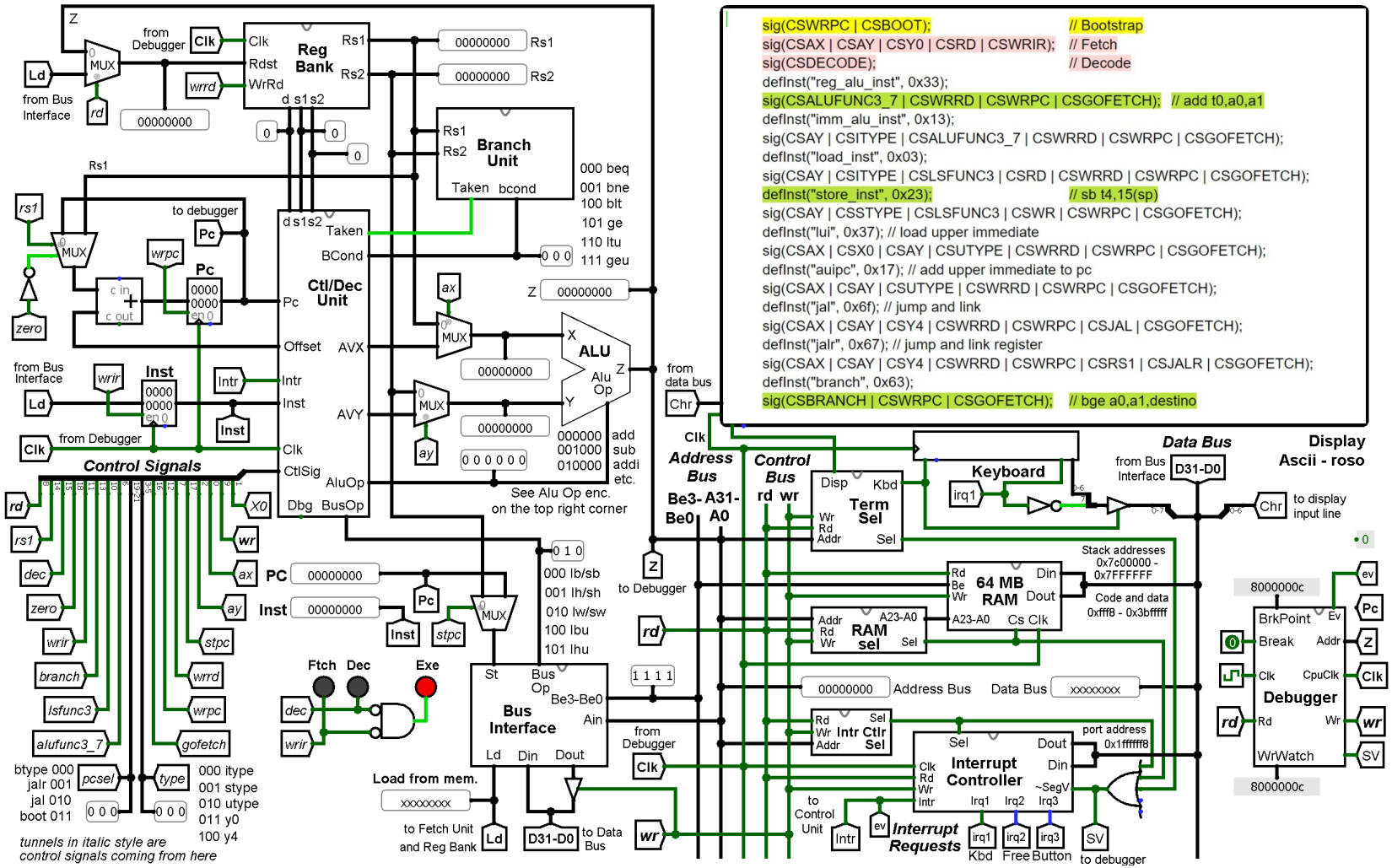
Metodología obligatoria: Lea la potencia y periodo actuales. Si el periodo actual es inferior al parámetro *period* significa que el motor necesita ir más lento y por lo tanto debe disminuir la potencia. Decremente en una unidad la potencia e invoque la función *sleep(1)* para dejar pasar un segundo. Debido a la inercia del motor, este puede demorar en responder al cambio de potencia. Repita hasta alcanzar el periodo requerido. Si se sobrepasa no importa, retorne igualmente. Para el caso en que el periodo actual es superior a *period*, debe aumentar la potencia. El procedimiento es análogo.

Pregunta 4

a) t5 ← Mem[s1]
b) s1 ← -14
c) t5 ← pc
d) Inst ← s1-14
e) t5 ← s1 - 14

Considere el diseño de LRV32IM microprogramado que usó en sus tareas. Al reverso del enunciado está el diagrama de bloques. El registro de instrucción *Inst* tiene cargada la instrucción *andi t5, s1, -14* (*t5* es el registro de destino). Evalúe de manera independiente si cada una de las 5 transferencias entre registros del cuadro de la derecha se puede realizar en *un solo ciclo del reloj*. Si la transferencia se puede realizar indique cuáles son la señales de control requeridas para llevarla a cabo. Si no se puede realizar, explique por qué.

CC4301 Arquitectura de Computadores – Examen – Primavera 2024 – Prof.: Luis Mateu




```
sig(CSWRPC | CSBOOT);                          // Bootstrap
sig(CSAX | CSAY | CSY0 | CSRD | CSWRIR);       // Fetch
sig(CSDECODE);                                 // Decode
defInst("reg_alu_inst", 0x33);
sig(CSALUFUNC3_7 | CSWRRD | CSWRPC | CSGOFETCH);  // add t0,a0,a1
defInst("imm_alu_inst", 0x13);
sig(CSAY | CSITYPE | CSALUFUNC3_7 | CSWRRD | CSWRPC | CSGOFETCH);
defInst("load_inst", 0x03);
sig(CSAY | CSITYPE | CSLSFUNC3 | CSRD | CSWRRD | CSWRPC | CSGOFETCH);
defInst("store_inst", 0x23);                   // sb t4,15(sp)
sig(CSAY | CSSTYPE | CSLSFUNC3 | CSWR | CSWRPC | CSGOFETCH);
defInst("lui", 0x37); // load upper immediate
sig(CSAX | CSX0 | CSAY | CSUTYPE | CSWRRD | CSWRPC | CSGOFETCH);
defInst("auipc", 0x17); // add upper immediate to pc
sig(CSAX | CSAY | CSUTYPE | CSWRRD | CSWRPC | CSGOFETCH);
defInst("jal", 0x6f); // jump and link
sig(CSAX | CSAY | CSY4 | CSWRRD | CSWRPC | CSJAL | CSGOFETCH);
defInst("jalr", 0x67); // jump and link register
sig(CSAX | CSAY | CSY4 | CSWRRD | CSWRPC | CSRS1 | CSJALR | CSGOFETCH);
defInst("branch", 0x63);
sig(CSBRANCH | CSWRPC | CSGOFETCH);            // bge a0,a1,destino
```

Alu Op encoding

bit 5 (left): mul/div operation
bit 4: immediate format
bit 3: substract / sra
bit 2-0 (right): **func3**

Alu Op func3 encoding

000 add/sub	100 xor
001 sll	101 srl/sra
010 slt	110 or
011 sltu	111 and

LRV32IM CPU

Risc-V 32 bit integer base instruction set with multiply/divide extension

Simple microprogrammed physical architecture

64 MB RAM

No Memory Management Unit

By Luis Mateu

Previous work by leijurv

GPL Licence