/\*\*\*\*\*module Rabbit\_2\_FPGA\_2\_DDS\_FIFO(init\_trigger, key\_1\_trigger, key\_3\_sweep, ten\_MHz\_ext, SDIO\_PE\_5, SCLK\_PE\_3,

SDIO, SCLK, IO\_UPDATE, DR\_CTL, OSK, CSB, DR\_HOLD, trigger,IO\_RESET, e\_number);\*\*\*\*\*testing\*\*\*\*\*/

module Rabbit\_2\_FPGA\_2\_DDS\_FIFO(init\_trigger, key\_1\_trigger, key\_3\_sweep, ten\_MHz\_ext, SDIO\_PE\_5, SCLK\_PE\_3,

SDIO, SCLK, IO\_UPDATE, DR\_CTL, OSK, CSB, DR\_HOLD, trigger,IO\_RESET, e\_number, sweep\_key\_flag, sweep\_end\_flag, complete\_end\_flag,

e\_flag,final\_massive\_stop, reset\_flag, init\_end\_flag, init\_key\_flag,i\_lsb,dont\_read\_flag, sweep\_trigger\_out);

input key\_3\_sweep;

input ten\_MHz\_ext;

input SDIO\_PE\_5;

input SCLK\_PE\_3;

input key\_1\_trigger;

input init\_trigger;

output reg SDIO;

output reg SCLK;

output reg IO\_UPDATE;

output reg DR\_CTL;

output reg OSK;

output reg CSB;

output reg DR\_HOLD;

output reg trigger;

output reg IO\_RESET;

output reg [4:0] e\_number;

output reg sweep\_trigger\_out;

integer J;

reg [31:0] i; //index for the memory as it is being read in

integer N; // index for the memory being read out

reg M; //tells when to send out the SDIO signal

reg L; //tells when to toggle the SCLK signal

reg [4:0] E;//counter for each full cycle

integer T;//time counter for sweeps

reg [10:0] B;//time counter for initialize

reg [8:0] P;//index for initialize

reg [9:0] G;//counter to time complete reset

reg Q;

reg R;

/\*\*\*\*\*\*for no-dwell)mode

parameter [0:231] INIT\_REG = {8'h0b,32'h0,32'h0,128'h000000400001004e002002353fc1c803,32'h0};\*\*\*\*\*/

//parameter [0:231] INIT\_REG = {8'h0b,32'h0,32'h0,128'h0000004000010048002002353fc1c803,32'h0};

//for non auto clear but clear

parameter [0:231] INIT\_REG = {8'h0b,32'h0,32'h0,8'h00,32'h00001000,8'h01,32'h00480020,8'h02,32'h353fc1c8,8'h03,32'h0};

//used to clear digital ramp accumluator

parameter [0:39] UNCLEAR\_REG = {8'h00,32'h00000000};

output reg reset\_flag;

output reg sweep\_key\_flag; // this is high during the time that the sweep info is being sent out to the DDS

//it must be reset every time that each signal has been sent

output reg sweep\_end\_flag;//this flag is: when it is set sweep\_key\_flag can be set back to zero

output reg complete\_end\_flag; //this flag will occur when the whole memory\_reg has been sent (when N>=i)

//it resets the N and i indices to zero so the thing can be restarted

output reg i\_lsb;

//This register is the most significant in the whole program. The program collects the data from the rabbit and stores

// it in this register and then reads the data from this register when sending it to the DDS.

//The register has 3680 bits because it is capable of storing 20 sweeps (20\*184=3680). However,

//not all of these bits will be used if storing less than 20 sweeps.

reg [0:3679] memory\_reg;

output reg init\_end\_flag;

output reg init\_key\_flag;

output reg e\_flag;

output reg final\_massive\_stop;

output reg dont\_read\_flag;

/\*\*\*\*\*\*\*testing\*\*\*\*

reg sweep\_key\_flag; // this is high during the time that the sweep info is being sent out to the DDS

//it must be reset every time that each signal has been sent

reg sweep\_end\_flag;//this flag is: when it is set sweep\_key\_flag can be set back to zero

reg complete\_end\_flag; //this flag will occur when the whole memory\_reg has been sent (when N>=i)

//it resets the N and i indices to zero so the thing can be restarted

reg [0:3679] memory\_reg;

reg init\_end\_flag;

reg init\_key\_flag;

reg e\_flag;\*\*\*\*\*\*\*\*\*\*testing\*\*\*/

initial

begin

i<=0; //index for the memory as it is being read in

N<=0; // index for the memory being read out

M<=0; //tells when to send out the SDIO signal

L<=0; //tells when to toggle the SCLK signal

T<=0;

B<=0;//time counter for initialize

P<=0;//index for initialize

E<=0;

reset\_flag<=0;

final\_massive\_stop<=0;

Q<=0;

R<=0;

e\_flag<=0;

end

//this section of code deals with making the decisions about when to start doing what

//its main job is to make sure the FPGA knows when to send the signal to the DDS

// and when to reset everything

always@ (negedge ten\_MHz\_ext)

begin

sweep\_trigger\_out<=IO\_UPDATE/\*!key\_3\_sweep\*/;

CSB <=0;//must be tied to zero to have DDS recieve

OSK <=0;

DR\_HOLD <=0;//must be tied to zero to let the DDS work smooth

//this sets the sweep\_key\_flag when the key\_3 is depressed and sweep\_key\_flag

// has not been set in 1 sec to "debounce" the keys. sweep\_end\_flag is

//set when sweep\_key\_flag has been set for 1 sec

if ((key\_3\_sweep!= 1) && (sweep\_end\_flag == 0))

begin

sweep\_key\_flag <=1;

end

if (sweep\_end\_flag != 0)

begin

sweep\_key\_flag <=0;

end

//this sets the init\_key\_flag when the rabbit sends the initalize signal

//don't know how init\_end\_flag should be used

if ((init\_trigger != 0) && (init\_end\_flag == 0))

begin

init\_key\_flag <=1;

end

if (init\_end\_flag != 0)

begin

init\_key\_flag <=0;

end

if (init\_trigger !=0)

begin

dont\_read\_flag<=1;

end

if (dont\_read\_flag!=0)

begin

J<=J+1;

end

if (J>=2000/\*70000\*/)

begin

J<=0;

dont\_read\_flag<=0;

end

//this section will set the complete\_end\_flag which will indicate that the whole set of sweep has been run

//this happens when N>=i this will reset everything important including setting N and i back to Zero

if ((N>=i)&& (sweep\_key\_flag !=0))

begin

complete\_end\_flag <=1;

end

if (N<i)

begin

complete\_end\_flag<=0;

end

if (final\_massive\_stop !=0)

begin

sweep\_key\_flag<=0;

end

end

//this section of code deals with sending out the signal to the DDS

always@ (posedge ten\_MHz\_ext)

begin

i\_lsb<=i[0];

/\*\*\*\*\*\*

Thi part of the code deals with the intialising parameters being sent to the DDS. This specifies the settings we need on the DDS.

the actual signal that must be sent to the DDS is specified above in the parameter,

"INIT\_REG = {8'h0b,32'h0,32'h0,128'h0000004000010048002002353fc1c803,32'h0}; "

The code below increments a counting register, B, which is used to time the signals being sent to

the following pins:

IO\_RESET

SDIO

SCLK

IO\_UPDATE

Note that there is no change in the DR\_CTL signal because that pin only deals with actual sweeps.

Also IO\_RESET is only used during this cycle

\*\*\*\*\*/

if (init\_key\_flag != 0)

begin

//B increments while init\_key\_flag is high. when B

B<=B+1'b1;

end

if (((B>= 568) && (B<573)) || ((B>=751) && (B<756)))

begin

IO\_UPDATE <= 1;

end

if ((((B<568) || (B>=573))&&((B<751)||(B>=756)))&&(sweep\_key\_flag == 0))

begin

IO\_UPDATE <=0;

end

if ((B>=2) && (B<7))

begin

IO\_RESET <= 1;

end

if ((B<2) || (B>=7))

begin

IO\_RESET <=0;

end

if ((B>=55) && (B<519))

begin

if(Q>=1)

begin

Q<=0;

SCLK<=1;

end

if(Q<1)

begin

Q<=Q+1'b1;

SCLK<=0;

end

if (R < 1)

begin

R<=R+1'b1;

SDIO <= INIT\_REG[P];

end

if (R >= 1)

begin

P<= P+1'b1;

R<=0;

end

end

if ((B>=622)&&(B<702))

begin

if(Q>=1)

begin

Q<=0;

SCLK<=1;

end

if(Q<1)

begin

Q<=Q+1'b1;

SCLK<=0;

end

if (R < 1)

begin

R<=R+1'b1;

SDIO<=UNCLEAR\_REG[P];

end

if (R>=1)

begin

R<=0;

P<= P+1'b1;

end

end

if ((((B<55) || (B>=519))&&((B<622)||(B>=702)))&&(sweep\_key\_flag == 0))

begin

P<=0;

SCLK<=0;

SDIO<=0;

end

if (B>=1000)//testing

begin

init\_end\_flag<=1;

B<=0;

if (final\_massive\_stop!=0)

begin

reset\_flag<=1;

end

end

if (B<1000)

begin

init\_end\_flag<=0;

end

if (i<=1)

begin

reset\_flag<=0;

end

if ((sweep\_key\_flag != 0)&&(final\_massive\_stop==0))

begin

//T increments while sweep\_key\_flag is high. when T >=20,000,000 sweep\_key\_flag will go back to zero

T<=T+1;

end

//when sweep\_end\_flag is high sweep\_key\_flag can be reset the number must be very large because the trigger from the keys can be >1sec

//this is the section of code that toggles the IO\_UPDATE

if ((T>= 422) && (T<423))

begin

IO\_UPDATE <= 1;

end

if (((T<422) || (T>=423))&&(init\_key\_flag == 0))

begin

IO\_UPDATE <=0;

end

//this is the section of code where the message is actually being sent

//when this section is not occuring SCLK and SDIO is low

if ((T>=5) && (T<373))

begin

if(L>=1)

begin

L<=0;

SCLK<=1;

end

if(L<1)

begin

L<=L+1'b1;

SCLK<=0;

end

if (M < 1)

begin

M<=M+1'b1;

//the main difference with this program to have multiple sweeps

//is that N is not reset after every sweep

SDIO <= memory\_reg[N];

end

if (M >= 1)

begin

N<= N+1'b1;

M<=0;

end

end

if (((T<5) || (T>=373))&&(init\_key\_flag == 0))

begin

SCLK<=0;

SDIO<=0;

end

if (key\_1\_trigger == 1)

begin

trigger <= 0;

end

if (key\_1\_trigger== 0)

begin

trigger <= 1;

end

if (T==20000000)

begin

sweep\_end\_flag <= 1;

T<=0;

E<=E+1'b1;

e\_number<=E;//test E for DR\_CTL trouble shoot

end

if (T<20000000)

begin

sweep\_end\_flag <=0;

end

if (e\_flag !=0)

begin

G<=G+1'b1;

end

if (G>=10'b1111111111)

begin

e\_flag<=0;

G<=0;

final\_massive\_stop<=1;

T<=0;

end

if (reset\_flag !=0)

begin

E<=0;

final\_massive\_stop<=0;

T<=0;

end

//after whole sweep routine it can start over

if (complete\_end\_flag !=0)

begin

e\_flag<=1;

N<=0;

end

if ((T>=422) && (T<430))

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=0;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=1;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

end

end

if (T>=430)

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=1;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=0;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

end

end

end

/\*

always@ (negedge ten\_MHz\_ext)

begin

if ((T>=422) && (T<425))

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=0;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=1;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

end

end

if (T>=425)

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=1;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=0;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

end

end

end\*/

///this code gives the correct DR\_CTL sequence based on whether the sweep is up or down

//it compares the appropriate places in the larger memory register within each 184\_bit section and then this section sets the

//DR\_CTL to its state after the toggle

// low to high sweeps occur when the first part of the step size register is higher than the second

//the lowering step size is the highest "memory\_reg[184\*n + 8: 184\*n + 39] > memory\_reg[184\*n + 40: 184\*n + 71]"

//where DR\_CTL is set. so from hi to low, 8:39<40:71, after toggle Dr\_CTL is cleared -- DR\_DTL to start high and go low.

/\*

if ((T>=420) && (T<430))

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=0;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=1;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

end

end

///this code gives the correct DR\_CTL sequence based on whether the sweep is up or down

//it compares the appropriate places in the larger memory register within each 184\_bit section and then this section sets the

//DR\_CTL to its state after the toggle

// low to high sweeps occur when the first part of the step size register is higher than the second

//the lowering step size is the highest "memory\_reg[184\*n + 8: 184\*n + 39] > memory\_reg[184\*n + 40: 184\*n + 71]"

//where DR\_CTL is set. so from hi to low, 8:39<40:71, after toggle Dr\_CTL is cleared -- DR\_DTL to start high and go low.

if (T>=430)

begin

if (E<=18)

begin

if (E<=17)

begin

if (E<=16)

begin

if (E<=15)

begin

if (E<=14)

begin

if (E<=13)

begin

if (E<=12)

begin

if (E<=11)

begin

if (E<=10)

begin

if (E<=9)

begin

if (E<=8)

begin

if (E<=7)

begin

if (E<=6)

begin

if (E<=5)

begin

if (E<=4)

begin

if (E<=3)

begin

if (E<=2)

begin

if (E<=1)

begin

if (E<=0)

begin

if (memory\_reg[8:39] > memory\_reg[40:71])

begin

DR\_CTL<=1;

end

if (memory\_reg[8:39] < memory\_reg[40:71])

begin

DR\_CTL<=0;

end

end

else

begin

if (memory\_reg[184+8:184+39] > memory\_reg[184+40:184+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184+8:184+39] < memory\_reg[184+40:184+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*2+8:184\*2+39] > memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*2+8:184\*2+39] < memory\_reg[184\*2+40:184\*2+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*3+8:184\*3+39] > memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*3+8:184\*3+39] < memory\_reg[184\*3+40:184\*3+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*4+8:184\*4+39] > memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*4+8:184\*4+39] < memory\_reg[184\*4+40:184\*4+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*5+8:184\*5+39] > memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*5+8:184\*5+39] < memory\_reg[184\*5+40:184\*5+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*6+8:184\*6+39] > memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*6+8:184\*6+39] < memory\_reg[184\*6+40:184\*6+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*7+8:184\*7+39] > memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*7+8:184\*7+39] < memory\_reg[184\*7+40:184\*7+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*8+8:184\*8+39] > memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*8+8:184\*8+39] < memory\_reg[184\*8+40:184\*8+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*9+8:184\*9+39] > memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*9+8:184\*9+39] < memory\_reg[184\*9+40:184\*9+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*10+8:184\*10+39] > memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*10+8:184\*10+39] < memory\_reg[184\*10+40:184\*10+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*11+8:184\*11+39] > memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*11+8:184\*11+39] < memory\_reg[184\*11+40:184\*11+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*12+8:184\*12+39] > memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*12+8:184\*12+39] < memory\_reg[184\*12+40:184\*12+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*13+8:184\*13+39] > memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*13+8:184\*13+39] < memory\_reg[184\*13+40:184\*13+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*14+8:184\*14+39] > memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*14+8:184\*14+39] < memory\_reg[184\*14+40:184\*14+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*15+8:184\*15+39] > memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*15+8:184\*15+39] < memory\_reg[184\*15+40:184\*15+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*16+8:184\*16+39] > memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*16+8:184\*16+39] < memory\_reg[184\*16+40:184\*16+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*17+8:184\*17+39] > memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*17+8:184\*17+39] < memory\_reg[184\*17+40:184\*17+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*18+8:184\*18+39] > memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*18+8:184\*18+39] < memory\_reg[184\*18+40:184\*18+71])

begin

DR\_CTL<=0;

end

end

end

else

begin

if (memory\_reg[184\*19+8:184\*19+39] > memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=1;

end

if (memory\_reg[184\*19+8:184\*19+39] < memory\_reg[184\*19+40:184\*19+71])

begin

DR\_CTL<=0;

end

end

end\*/

// end

//this section of code deals with reading the message from the rabbit

always@ (posedge SCLK\_PE\_3)

begin

if (reset\_flag !=0)

begin

i<=0;

memory\_reg<= 3680'b0;

end

if (dont\_read\_flag == 0)

begin

memory\_reg[i] <= SDIO\_PE\_5;

i<=i+1;

end

end

endmodule