DICD Coursework

Bitslice Datapath Design

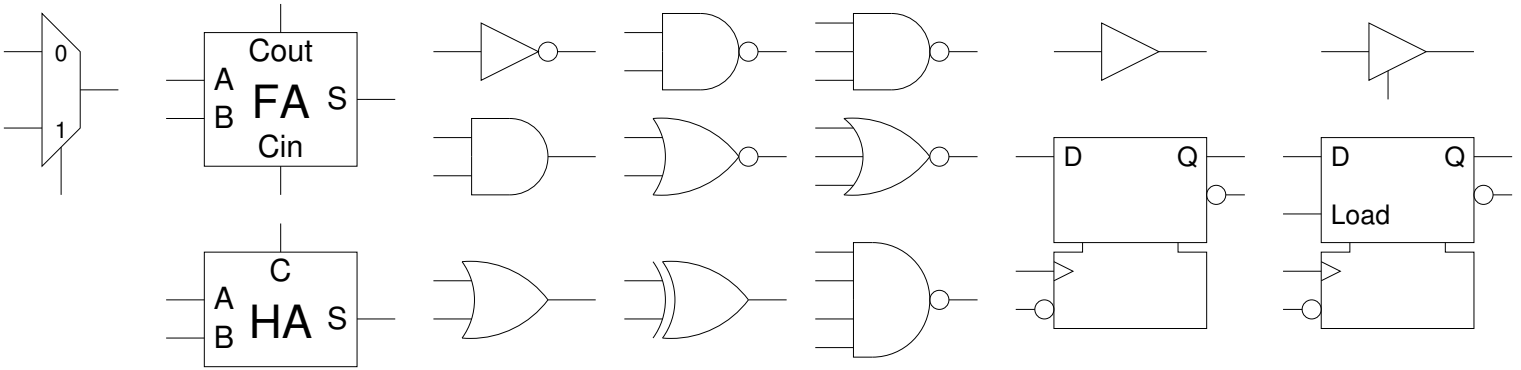
- Working individually.
- Implement an 8 bit divider datapath using bitslice techniques.
- Implement a synthesizable SystemVerilog control unit for your datapath.

Three options:

1. Datapath Architecture and Algorithm Provided, Control Signals Specified
2. Datapath Architecture and Algorithm Provided
3. Nothing Provided

Bitslice Datapath Design

Cell library[1]



- Combinational Cells

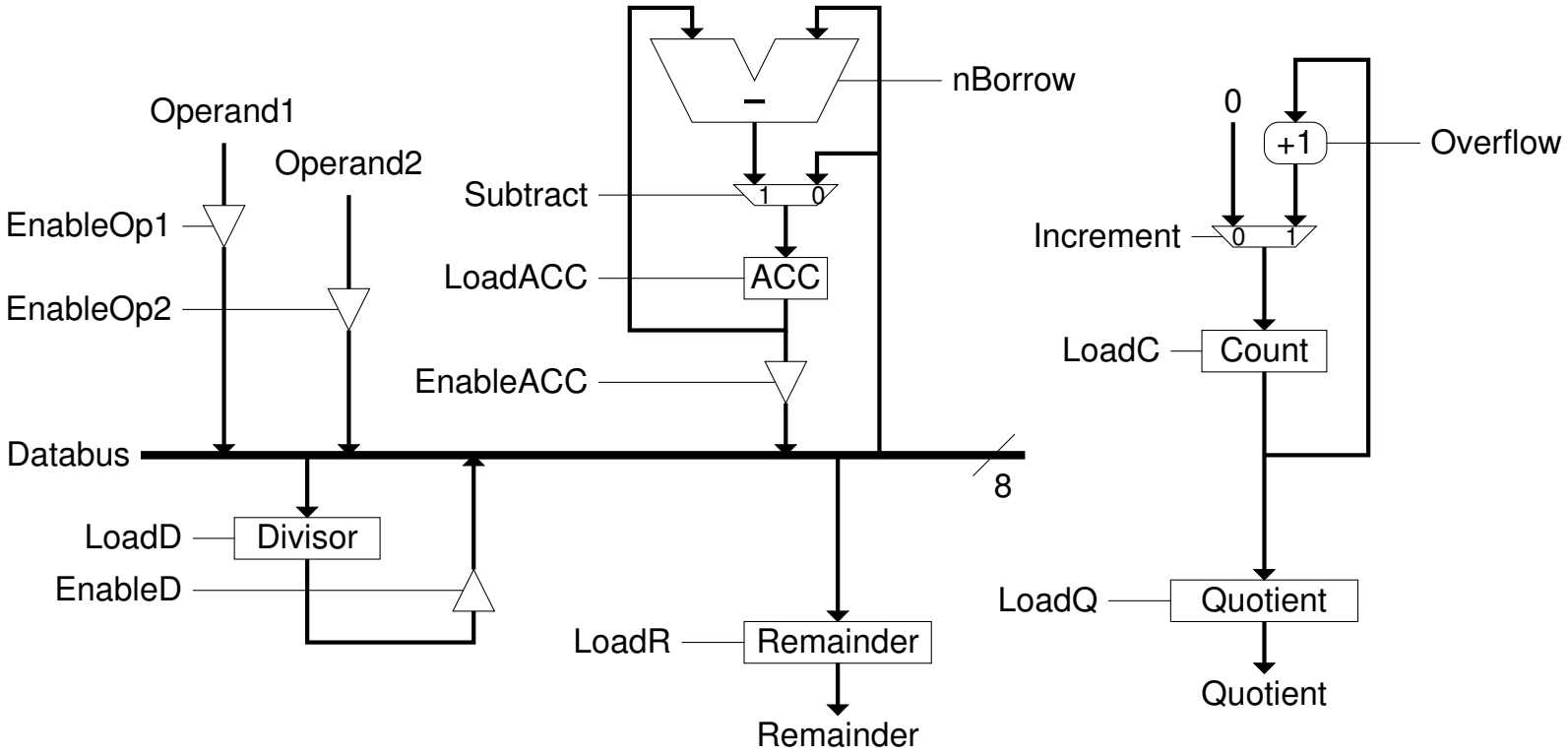
```
mux2 fulladder halfadder inv
and2 or2 nand2 nor2 xor2
nand3 nor3 nand4
buffer trisbuf
```

- Sequential Cells

```
scandtype scanreg
```

[1]Please use the symbols shown when representing these cells in gate level schematics. Test and SDI connections have been intentionally omitted.

8 bit Divider Datapath - TYPE#1



Datapath Architecture

Division Algorithm - TYPE#1

- The datapath has been designed to implement the following simple division algorithm:

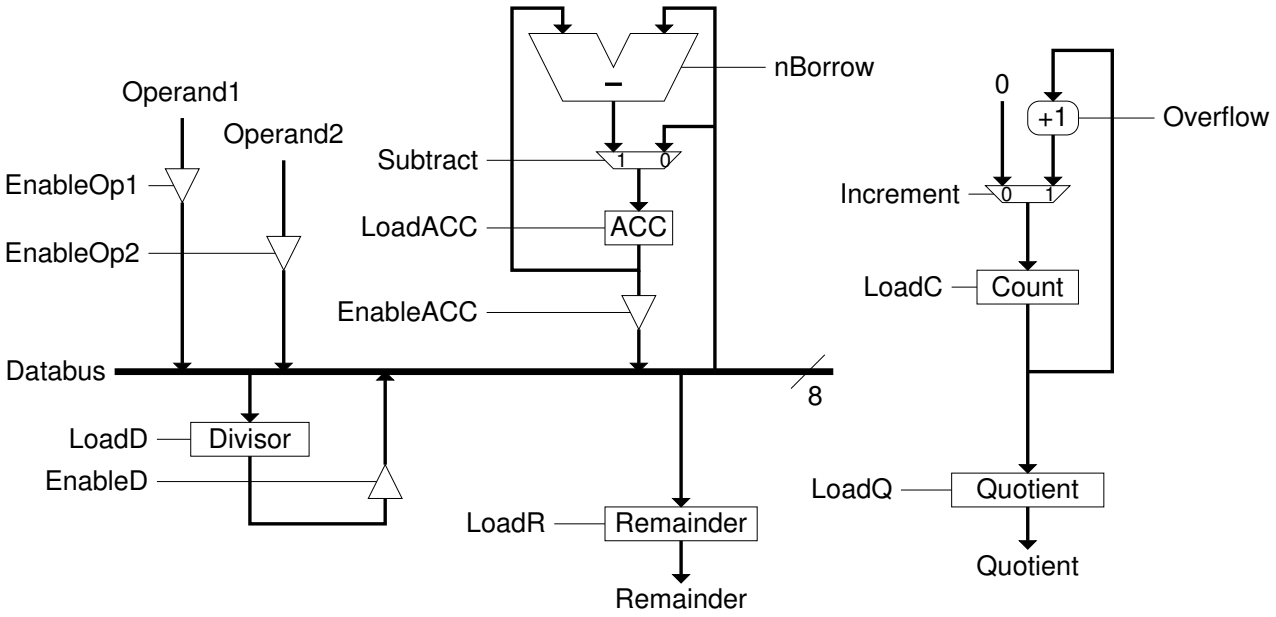
```
Count = 0;
ACC = Operand1;
Divisor = Operand2;

while ( ACC >= Divisor )
  begin
    Count = Count + 1;
    ACC = ACC - Divisor;
  end

Remainder = ACC;
Quotient = Count;
```

Bitslice Datapath Design

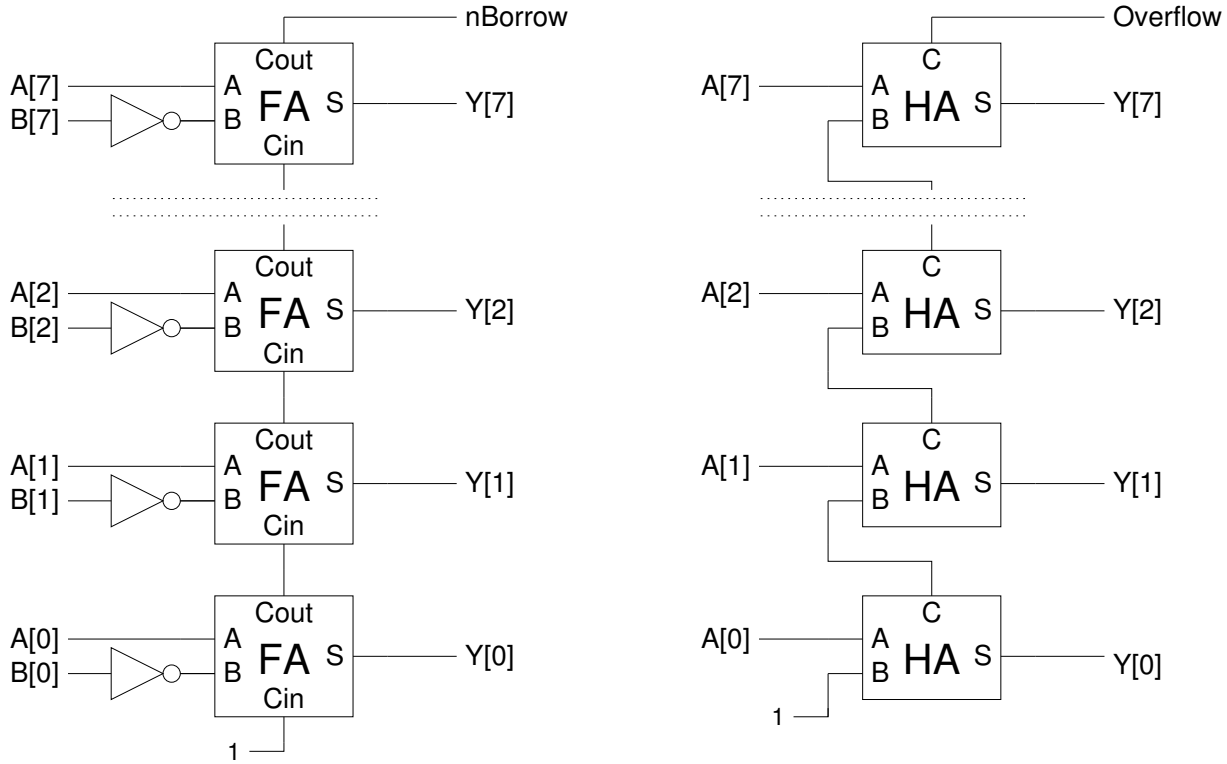
Implementaion of Blocks



- Each 8 bit register is implemented as 8 scannable register cells. The signals Clock, nReset, Test etc are not shown to keep the datapath diagram simple.
- The 8 bit subtracter and incrementer are implemented as ripple carry circuits based around full adders and half adders.

Bitslice Datapath Design

- The subtracter and incrementer[2] circuits:



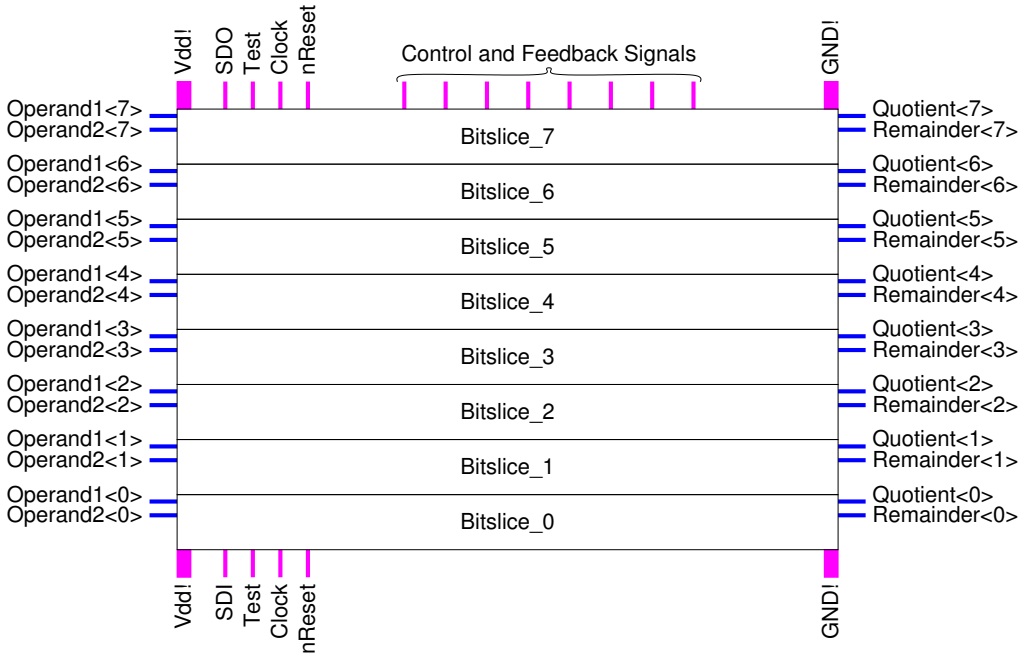
Note that the `Overflow` signal from the incrementer is not needed by the algorithm but can be used to detect *divide by zero*.

[2] a half adder for the incrementer can be constructed from an `xor2` cell and an `and2` cell if no `halfadder` cell is available

Bitslice Datapath Design

8 Identical Bitslice Cells

- all inter-bitslice wiring is by butting.



Datapath Cell

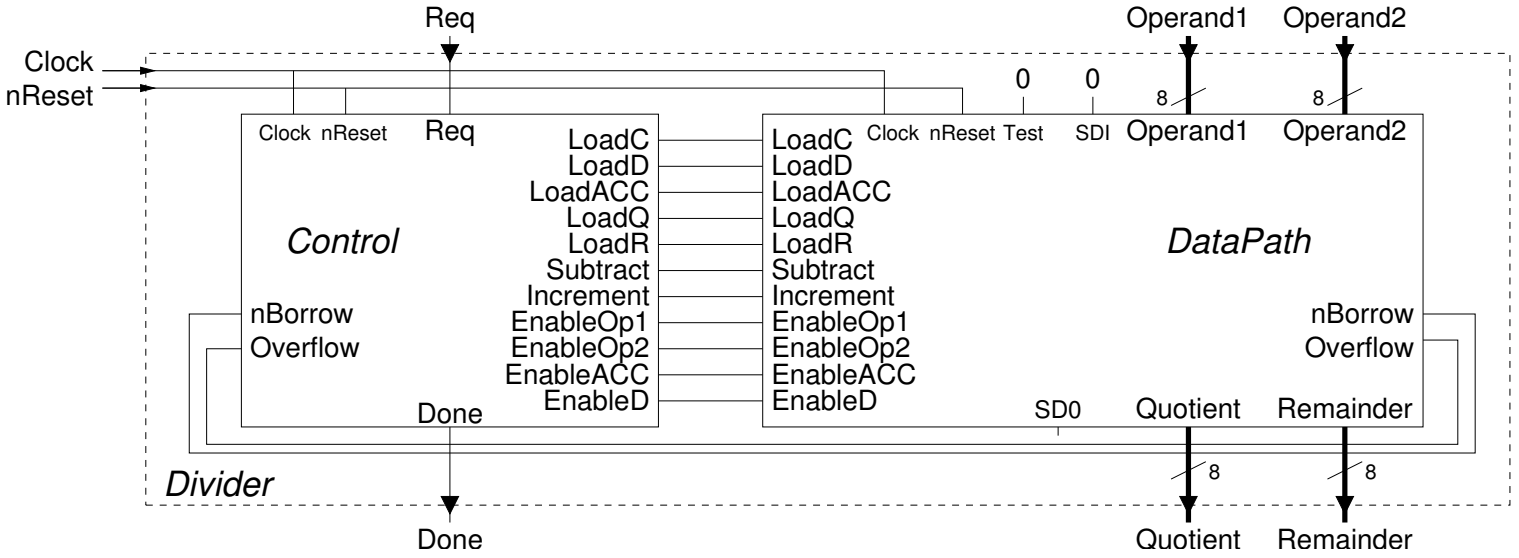
inputs: Clock, nReset, Test, SDI, Operand1, Operand2, Subtract, Increment, LoadC, LoadD, LoadACC, LoadQ, LoadR, EnableOp1, EnableOp2, EnableACC, EnableD

outputs: SDO, Quotient, Remainder, nBorrow, Overflow

Bitslice Datapath Design – SystemVerilog

- **control** – synthesizable behavioural model generates control signals
- **divider** – structural model instances **datapath** and **control**
- **divider_stim** – test harness.

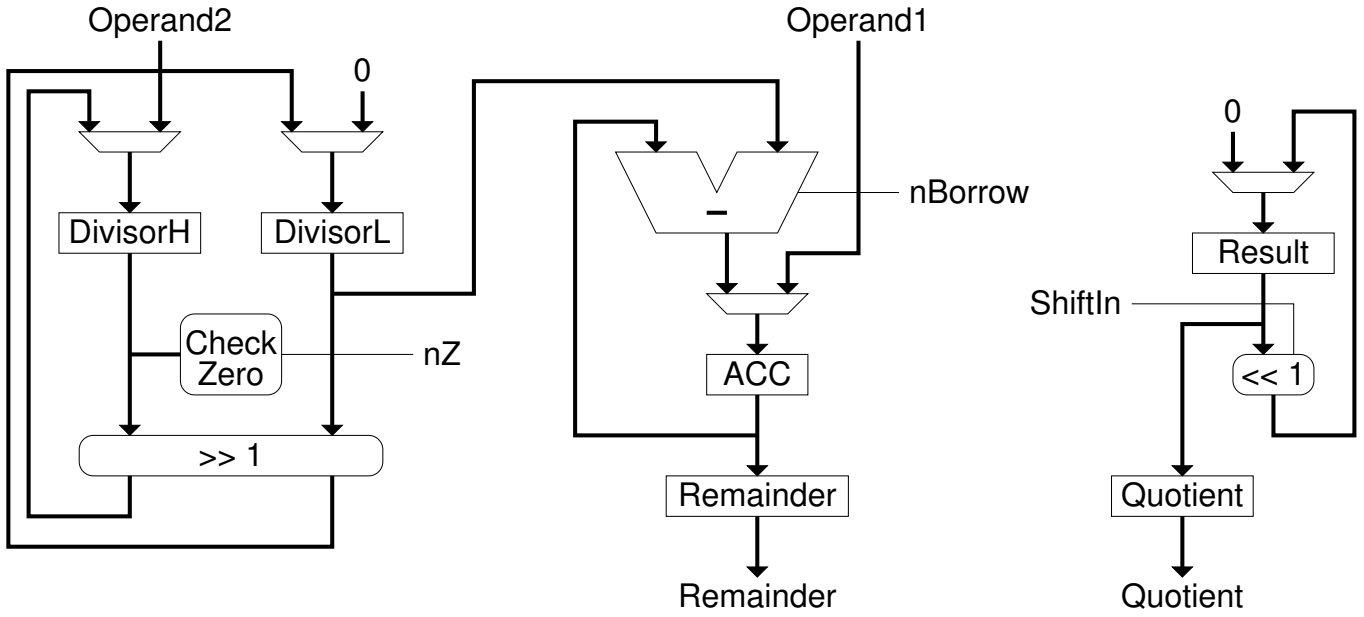
  To indicate that a new division is required, the stimulus will raise the `Req` signal for one clock cycle. Once the `Quotient` and `Remainder` become available, the **control** module will raise the `Done` signal for one cycle[3].



[3]The `Quotient` and `Remainder` values must remain stable and valid between `Done` cycles

8 bit Divider Datapath - TYPE#2

The following datapath[4] allows the implementation of a more efficient algorithm:



You can try this alternative (TYPE#2) datapath if you want to show off your talents.

[4]control signals are not specified here to allow greater flexibility in the design (i/o for complete divider will be unchanged – Req, Done, Operand1, Operand2, Quotient, Remainder, Clock, nReset)

Division Algorithm - TYPE#2

```
ACC = Operand1; Result = 0;
DivisorH = Operand2; DivisorL = 0;

for ( i = 0; i < 8; i = i+1 )
  begin
    {DivisorH,DivisorL} = {DivisorH,DivisorL} >> 1;
    Result = Result << 1;
    if ( (DivisorH == 0) && ( ACC >= DivisorL) )
      begin
        ACC = ACC - DivisorL;
        Result[0] = 1;
      end
  end

Remainder = ACC;
Quotient = Result;
```

Warning - think before making your life harder. With greater flexibility comes a greater chance to make mistakes. – A good working implementation of a TYPE#1 system will get more marks than an almost working implementation of a TYPE#2 system.

8 bit Divider Datapath - TYPE#3

Division Algorithm - TYPE#3

Other algorithms and datapaths may be acceptable but only if they are passed by me for the use of individual students.

Location of files

- Cell Library Directory

  All magic leaf cells should exist **only** in the cell library directory:

  `~/design/magic/c035u/cell_lib`

- Bitslice Directory

  The two magic hierarchical cells `bitslice.mag` & `datapath.mag` together with all of the SystemVerilog files should exist **only** in the bitslice directory:

  `~/design/magic/c035u/bitslice`

To enable access to the leaf cells when creating a hierachical cell you will need to use the magic addpath command:

`:addpath /home/<username>/design/magic/c035u/cell_lib`

before using the `:getcell` command.
The easiest way to do this is to include the appropriate addpath command in a ".magic" file in the bitslice directory. This is done automatically by the startup script:

`init_bitslice_directories`

Submission – Files

Prepare script (`prepare_dicd desex4`) will collect the folowing files:

- `leftbuf.mag rightend.mag` + `fulladder.mag` ... (all leaf cells used)
- `bitslice.mag bitslice.vnet bitslice.sv bitslice_stim.sv`
- `datapath.mag datapath.vnet datapath.sv datapath_stim.sv`
- `control.sv control_stim.sv`
- `divider.sv divider_stim.sv divider.tcl`

Submission – Supporting Figures

System design diagrams (in each case **all** control signals must be shown):

- Datapath Architecture Diagram — *single A4 page PDF document readable at A5 size*
- Bitslice Gate Level Schematic — *single A4 page PDF document readable at A5 size*
- ASM Chart — *single A4 page PDF document readable at A5 size*
- All 3 figures combined on one page — *single A4 page PDF document*

Deadline for both submissions is 18:00 on Wednesday 22nd January.