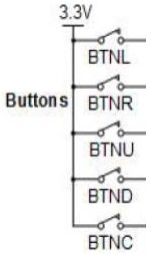
Picoblaze_GCD

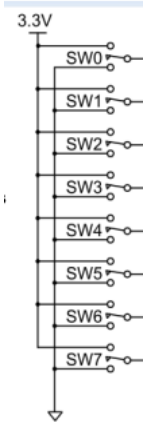
GCD on Picoblaze

Implementation, Simulation, and Synthesis

Files and Instances for synthesis

gcd_picoblaze.xdc





FPGA Artix 7 ee354_GCD_picoblaze_top.v module ee354_GCD_top

Input ports related fabric logic

Standard two instances of the picoblaze processor

Instance name
processor

picoblaze
processor
kcpsm6.v

module
kcpsm6

address[11:0]

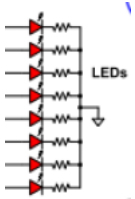
instruction[17:0]

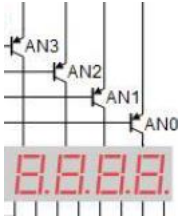
Instance name
program_rom

Program memory
BRAM
prom_gcd.v
produced from
prom_gcd.psm

module
prom_gcd

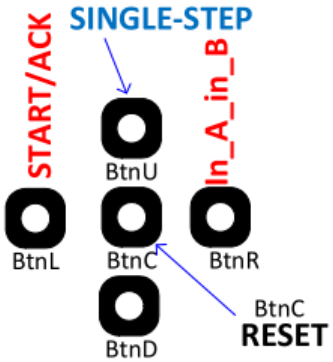
Output ports related fabric logic and standard SSD scanning logic

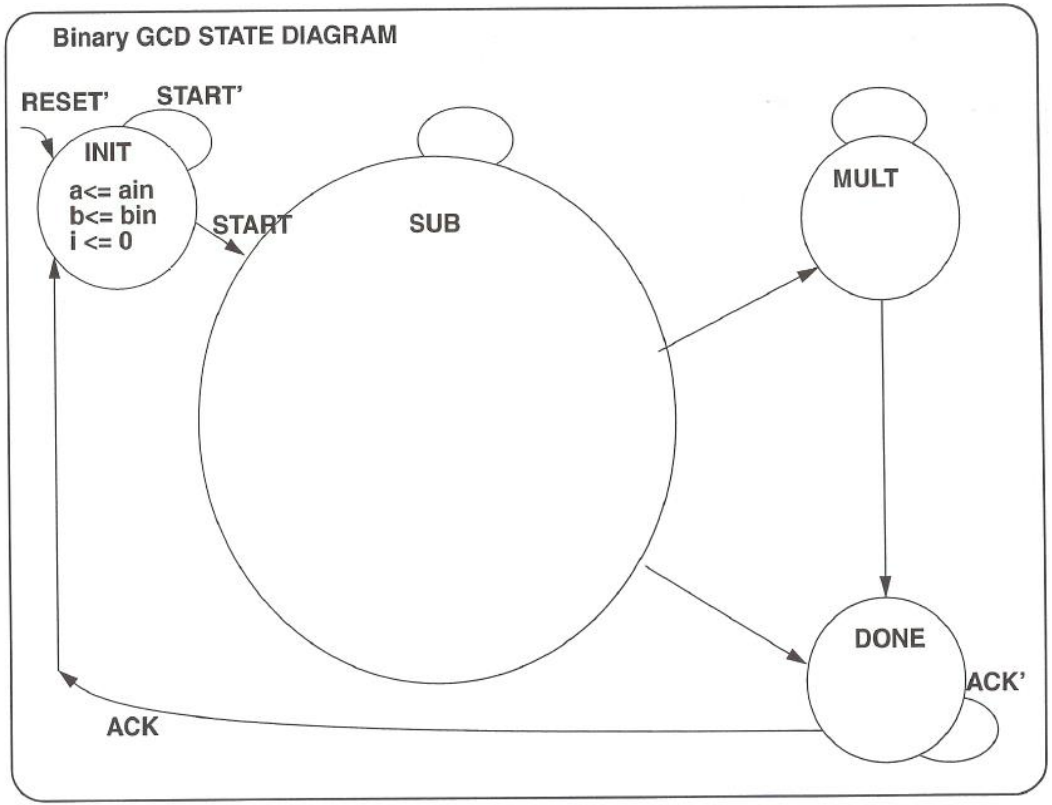




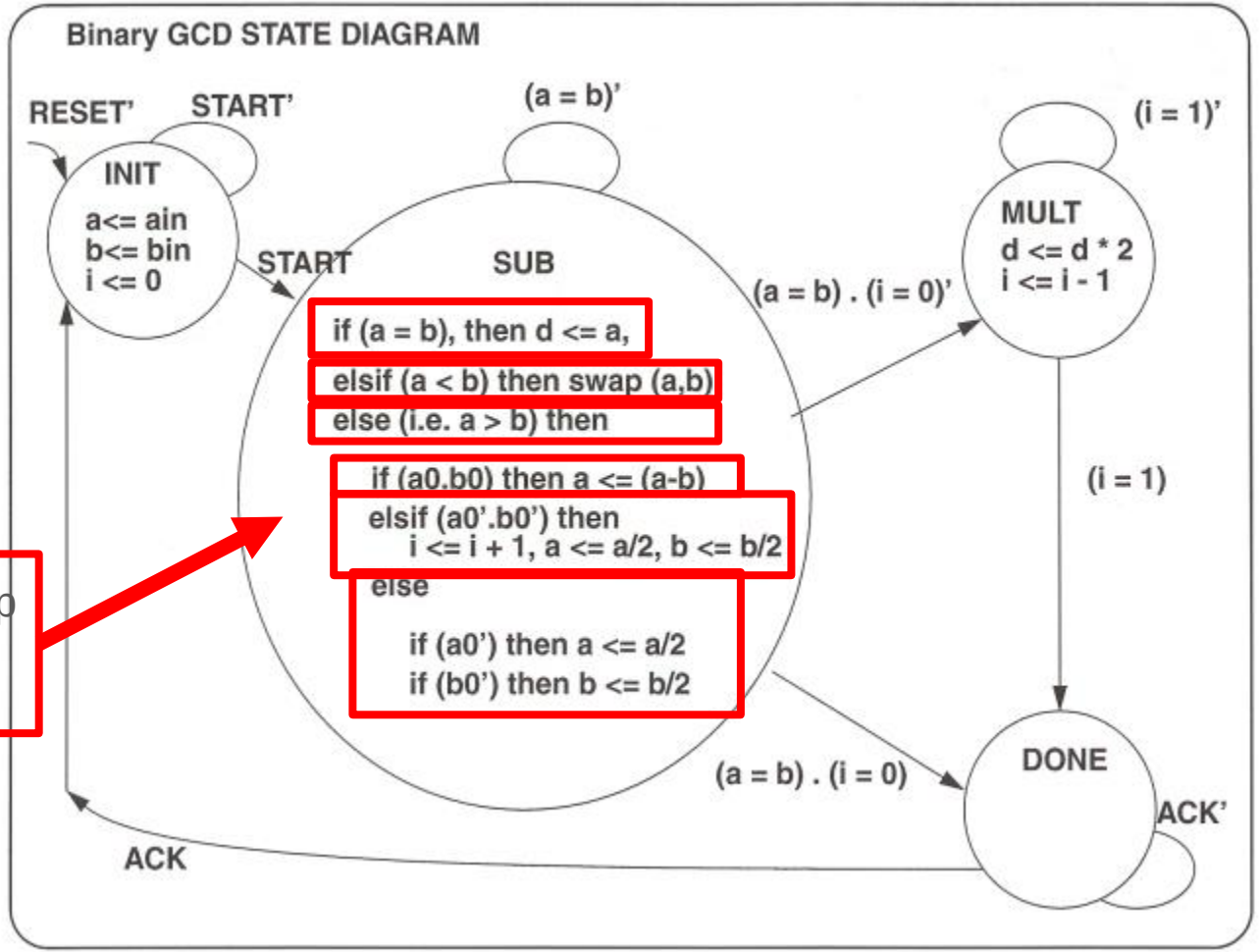
FSM in Assembly

- We need to program each state separately in assembly
- Start, Step and Ack are the external Control signals used here to exercise sequencing control.





FSM in Assembly



Each if/else represents a jump (conditional/unconditional)

The Carry (C) and the Zero (Z) flags

Logical

```
56  02xy0   AND sX, sY
56  03xkk   AND sX, kk
57  04xy0   OR sX, sY
57  05xkk   OR sX, kk
58  06xy0   XOR sX, sY
58  07xkk   XOR sX, kk
```

Arithmetic

```
59  10xy0   ADD sX, sY
59  11xkk   ADD sX, kk
60  12xy0   ADDCY sX, sY
60  13xkk   ADDCY sX, kk
61  18xy0   SUB sX, sY
61  19xkk   SUB sX, kk
62  1Axy0   SUBCY sX, sY
62  1Bxkk   SUBCY sX, kk
```

Test and Compare

```
63  0Cxy0   TEST sX, sY
63  0Dxkk   TEST sX, kk
64  0Exy0   TESTCY sX, sY
64  0Fxkk   TESTCY sX, kk
65  1Cxy0   COMPARE sX, sY
65  1Dxkk   COMPARE sX, kk
66  1Exy0   COMPARECY sX, sY
66  1Fxkk   COMPARECY sX, kk
```


It is important to understand how the Arithmetic, Logical, Test, and Compare instructions change the Carry (C) and the Zero (Z) flags and further how the conditional jumps, conditional calls, and conditional returns utilize the Carry (C) and the Zero (Z) flags.

Jump

```
87  22aaa   JUMP aaa
88  32aaa   JUMP Z, aaa
88  36aaa   JUMP NZ, aaa
88  3Aaaa   JUMP C, aaa
88  3Eaaa   JUMP NC, aaa
89  26xy0   JUMP@ (sX, sY)
```

Subroutines

```
92  20aaa   CALL aaa
93  30aaa   CALL Z, aaa
93  34aaa   CALL NZ, aaa
93  38aaa   CALL C, aaa
93  3Caaa   CALL NC, aaa
94  24xy0   CALL@ (sX, sY)
96  25000   RETURN
97  31000   RETURN Z
97  35000   RETURN NZ
97  39000   RETURN C
97  3D000   RETURN NC
```

Carry, Borrow, and Odd Parity

Even though the name of the flag is “Carry”, its meaning changes depending on the context.

Logical operations (AND, OR, and XOR) will reset the carry to zero.
The ADD operation will set the carry if the out-going carry C8 is true (otherwise will reset it).

The COMPARE operation performs comparison by performing subtraction. The COMPARE and SUBtract operations will set the carry if the subtrahend is bigger (otherwise will reset the carry). So the carry flag can be viewed as representing
the out-going borrow in subtraction.

In the case of TEST instructions, AND operation is performed to see if the result is zero or if the result has odd number of 1’s as indicated by the carry flag. So the carry flag can be viewed as an Odd Parity of the result 8 bits in the case of the TEST (and an Odd Parity of the result 8 bits plus one incoming carry bit in the case of the TESTCY).

Top design - fabric logic to interface with the PicoBlaze Processor

- How is our top design interfacing with the picoblaze processor through the INPUT and OUTPUT instructions? Let’s consider the INPUT instruction.

  INPUT sX, pp

  “pp” is the 8-bit port address in hex that the processor will output on port_id

  sX is one of the 16 8-bit registers (s0 through sf) inside the processor that will store the data coming through in_port[7:0]

In this lab, the processor informs the fabric logic “in which state it is currently at (Init, Sub, Mult, Done)” using the OUTPUTK instruction.

Possible to do so in two ways:
using OUTPUT sX, pp or using OUPUTK kk, p

Suppose constant 02 (kk = 02) needs to conveyed to the output port 01 (pp = 01)

```
LOAD s5, 02 ;
OUTPUT s5, 01
```

```
OUTPUTK 02, 01
```

```
; extract from our .psm file
state_initial: OUTPUTK 00000001'b, Current_State          ; Indicating Current State as Initial State
```

out_port interface

- We have two output instructions available to us, OUTPUT and OUTPUTK

  The regular OUTPUT instruction lets us output the 8-bit data of a register:

  OUTPUT sX, pp

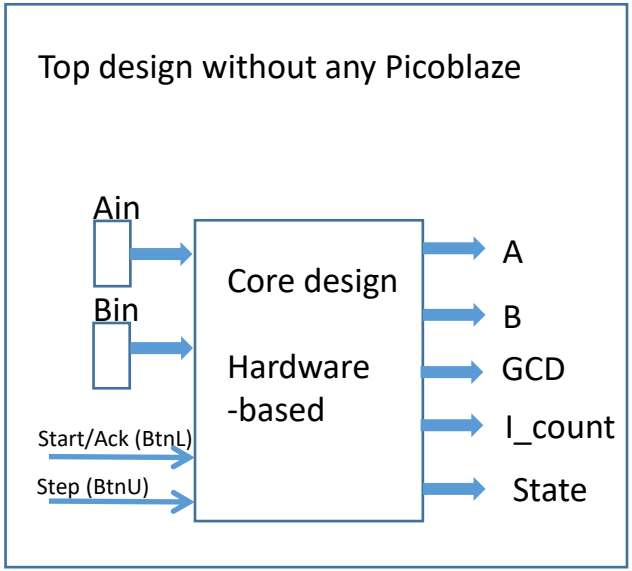
  sX is the register containing the data we are outputting, pp is the 8-bit address we output on port_id, write_strobe signal goes active.

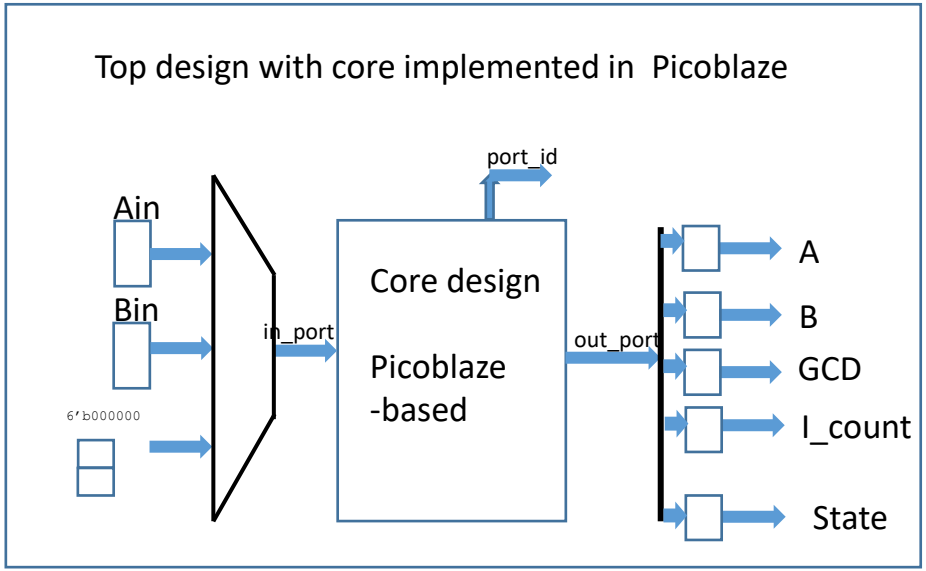
  The OUTPUTK instruction lets us output an 8-bit constant:

  OUTPUTK kk, p

  “kk” is the 8-bit constant we are outputting, p is the 4-bit address we output on port_id , k_write_strobe signal goes active.

Comparison of interface between top and the core designs in the case of the two designs

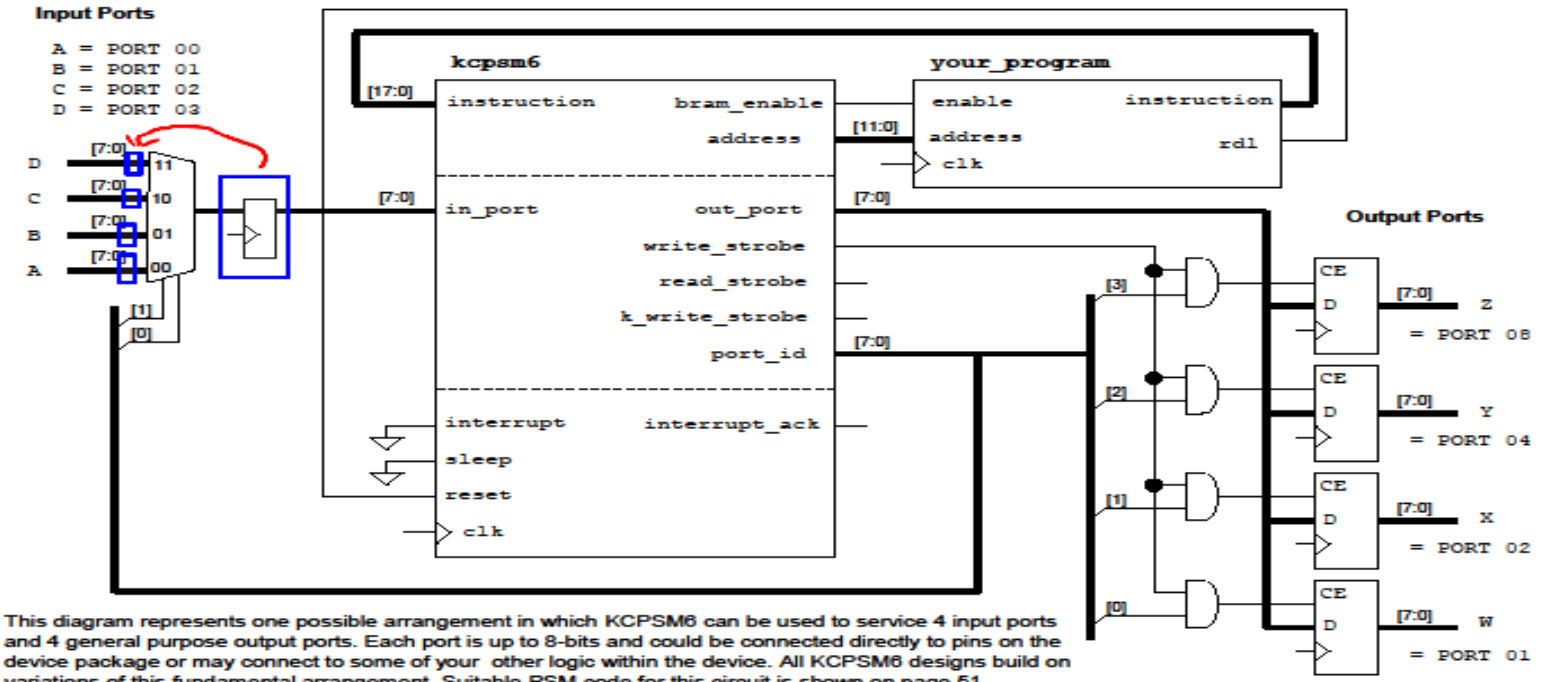




General Purpose I/O Ports

This design has been used to evaluate the maximum clock rates that can be achieved for a variety of device types and speed grades...

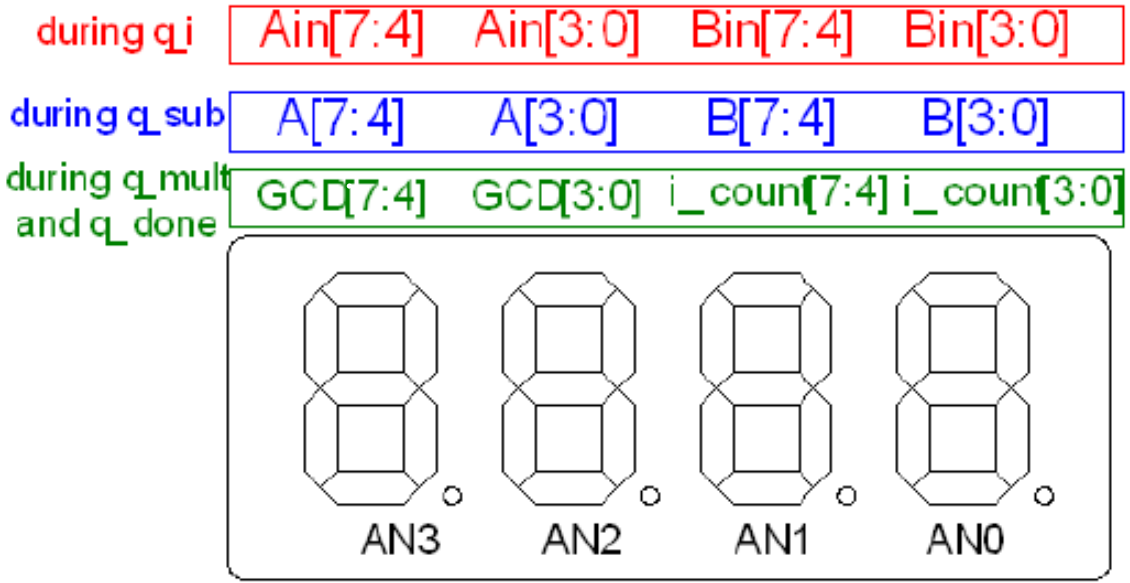
Spartan-6:	(-1L) ~82MHz	(-2) ~105MHz	(-3) ~136MHz
Virtex-6:			(-3) ~238MHz
Kintex-7:	(-1) ~185MHz		(-3) ~238MHz
Virtex-7:			(-3) ~232MHz



This diagram represents one possible arrangement in which KCPSM6 can be used to service 4 input ports and 4 general purpose output ports. Each port is up to 8-bits and could be connected directly to pins on the device package or may connect to some of your other logic within the device. All KCPSM6 designs build on variations of this fundamental arrangement. Suitable PSM code for this circuit is shown on page 51.

XILINX

SSD Display


during q_i
Ain[7:4] Ain[3:0] Bin[7:4] Bin[3:0]
during q_sub
A[7:4] A[3:0] B[7:4] B[3:0]
during q_mult
and q_done
GCD[7:4] GCD[3:0] i_coun[7:4] i_coun[3:0]
AN3
AN2
AN1
AN0

After completing your .psm file and TOP Verilog file, synthesize the top, download the .bit file and test.

You should be able to replicate the steps on the side reproduced from your ee354_GCD_rev5.pdf

```
1.  on reset                          0 0   0 0   Ain, Bin
2.  set sw[7:0]= 00111100
    and press BtnR (in_AB_pulse)      3 C   0 0   Ain, Bin
3.  set sw[7:0]= 01010100
    and press BtnR (in_AB_pulse)      3 C   5 4   Ain, Bin
Until now we are in q_i state and LD7 is on. Observe
that we now move to q_sub state and the LD6 glows.
4.  press BtnL (start_ack)
    to start                          3 C   5 4   A, B
5.  press BtnU to single-step         5 4   3 C   A, B swapped
6.  press BtnU to single-step         2 A   1 E   A, B halved, i = 1
7.  press BtnU to single-step         1 5   0 F   A, B halved, i = 2
8.  press BtnU to single-step         0 6   0 F   A-B, B
9.  press BtnU to single-step         0 F   0 6   A, B swapped
10. press BtnU to single-step         0 F   0 3   A, B/2
11. press BtnU to single-step         0 C   0 3   A-B, B
12. press BtnU to single-step         0 6   0 3   A/2, B
13. press BtnU to single-step         0 3   0 3   A/2, B
Until now we are in q_sub state and LD6 is on.
Observe that we now move to q_mult state and the LD5 glows.
14. press BtnU to single-step         0 3   0 2   GCD, i_count
15. press BtnU to single-step         0 6   0 1   GCD, i_count
Until now we are in q_mult state and LD5 is on.
Observe that we now move to q_done state and the LD4 glows.
16. press BtnU to single-step         0 C   0 0   GCD, i_count
17. press BtnU to single-step <= nothing new happens as
    the state machine is waiting for ACK.
18. press BtnL (start_ack)
    to ACK                            3 C   5 4   Ain, Bin
Now we are back in q_i state and LD7 glows.
```

5.1 Demonstrate to your TA/Mentor

Submit files on Unix as per the following posting on the Bb on the next page

5.2 Blackboard posting and Files for submission

Picoblaze_GCD

Using your experience with your previous assignment, where you designed, simulated, and implemented a Picoblaze-based 8-bit divider, here you will complete a Picoblaze-based GCD finder. You have completed already your gcd_verilog lab (non-picoblaze lab, involving designing the GCD RTL state machine in Verilog).

Directory: https://ece-classes.usc.edu/ee254/ee254l_lab_manual/PicoBlaze/Picoblaze_GCD

Assignment pdf: Picoblaze_GCD_handout.pdf
Videos (to be added next semester)

A .zip file to be downloaded and extracted into C:\Xilinx_projects:
An incomplete 8-bit divider design: Picoblaze_GCD.zip
The zip file contains a TA's completed .bit file (with dot points glowing on SSDs).

General reference: PicoBlaze/Picoblaze_Design_Steps_Demo_README_r1.pdf

Please demonstrate your completed Picoblaze_GCD design to your TA.
Submit your files to the class Unix account ee201@viterbi-scf1.usc.edu or ee201@viterbi-scf2.usc.edu using the following submit command

```
submit -user ee201 -tag Picoblaze_GCD ee354_GCD_picoblaze_top.v prom_gcd.psm names.txt
```