ECA 2 Assignment: Stack Processor

Gabiel Baraldi Souza (s3751163)
Veselin Daskalov (s2150883)

January 18, 2026

1 Exc1

```
system :: HiddenClockResetEnable dom
    => Signal dom (Unsigned 1) -> Signal dom (Unsigned 1)
system x = output
    where
        regout = register 0 xorout
        output = not' <$> xorout
        xorout = xor' <$> regout <*> x
```

- testSystem1: [1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0]
- testSystem2: [0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1]

2 Exc2

2.1

```
stackController :: State -> Instr  -> (State, Output)
stackController sp instr = case instr of
  Push v -> (sp + 1, (sp - 1, Just (sp, v)))
  _      -> (sp, (sp - 1, Nothing))
```

```
stackBlock :: HiddenClockResetEnable dom
  => Signal dom Instr -> Signal dom Output
stackBlock = mealy stackController 0
```

```
system :: HiddenClockResetEnable dom
  => Signal dom Instr -> Signal dom Value
system instr = output
  where
    output = blockRAMblock readAddr writeAddrCont
    (readAddr, writeAddrCont) = unbundle $ stackBlock instr
```

- testSystem

```
0.0
0.0
0.0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
```

- testStackBlock

```
(7,Nothing)
(7,Nothing)
(7,Just (0,10.0))
(0,Just (1,11.0))
(1,Just (2,12.0))
(2,Nothing)
(2,Nothing)
(2,Nothing)
(2,Nothing)
(2,Just (3,1.0))
(3,Nothing)
```

2.2

```
stackController :: State -> Instr  -> (State, Output)
stackController sp instr = case instr of
  Push v -> (sp + 1, (sp - 1, Just (sp, v)))
  Pop    -> (sp - 1, (sp - 1, Nothing))
  _      -> (sp, (sp - 1, Nothing))
```

- testSystem

```
0.0
0.0
0.0
0.0
10.0
11.0
12.0
11.0
10.0
0.0
0.0
0.0
1.0
```

- testStackBlock

```
(7,Nothing)
(7,Nothing)
(7,Just (0,10.0))
(0,Just (1,11.0))
(1,Just (2,12.0))
(2,Nothing)
(2,Nothing)
(2,Nothing)
(2,Nothing)
(2,Just (3,1.0))
(3,Nothing)
```

3 Exc3

3.1

```
processor :: () -> (Instr, Value) -> ((), Stack.Instr)
processor state (instr, value) = case instr of
  Push v -> ((), Stack.Push v)
  _      -> ((), Stack.Nop)
```

```
procBlock :: HiddenClockResetEnable dom
  => Signal dom (Instr, Value) -> Signal dom (Stack.Instr)
procBlock = mealy processor ()
```

```
system :: HiddenClockResetEnable dom
  => Signal dom Instr -> Signal dom Value
system instr = value
  where
    value = Stack.system stackInstruction
    stackInstruction = procBlock $ bundle (instr, value)
```

- testSystem

```
0.0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
```

3.2

3.3

```
processor :: State -> (Instr, Value) -> (State,
                                         (Stack.Instr, Maybe Value))
processor state (instr, value) = case state of
  Idle -> case instr of
    Push v  -> (Idle, (Stack.Push v, Nothing))
    Calc opCode -> (Calcing Nothing opCode, (Stack.Pop, Nothing))
    _       -> (Idle, (Stack.Nop, Nothing))
  Calcing storedValue opCode -> output
    where
      operation = case opCode of
        Add  -> (+)
        Mult -> (*)
      output = case storedValue of
        Just previous -> (Idle, (Stack.Push opResult, (Just opResult)))
            where opResult = operation value previous
        Nothing       -> (Calcing (Just value) opCode,
            (Stack.Pop, Nothing))
```

```
procBlock :: HiddenClockResetEnable dom
  => Signal dom (Instr, Value) -> Signal dom (Stack.Instr, Maybe Value)
procBlock = mealy processor Idle
```

```
system :: HiddenClockResetEnable dom
  => Signal dom Instr -> Signal dom (Maybe Value)
system instr = opResult
  where
    readResult = Stack.system stackInstruction
    (stackInstruction, opResult) = unbundle
        $ procBlock
        $ bundle (instr, readResult)
```

- testSystem

```
[
    Nothing,
    Nothing,
    Nothing,
    Nothing,
    Nothing,
    Nothing,
    Just 9.0,
    Nothing,
    Nothing,
    Nothing,
    Just 18.0,
    Nothing,
    Nothing,
    Nothing,
    Just 0.0
]
```

4 Exc4

```
fetcher :: State -> Proc.Instr -> (State, Output)
fetcher (pc, stallt) instr
    | stallt > 0                = ((pc, stallt - 1), (Proc.Nop, pc))
    | otherwise                 = ((pc + 1, delay), (instr, pc + 1))
        where
            delay = case instr of
                Proc.Calc _ -> 2
                _           -> 0
```

```
instrBRAM :: HiddenClockResetEnable dom =>
    Signal dom Address -> Signal dom
        (Maybe (Address, Proc.Instr)) -> Signal dom Proc.Instr
instrBRAM = blockRam $ testProgram
```

```
system :: HiddenClockResetEnable dom => Signal dom Proc.Instr
system = procInstr
    where
        (procInstr, addr) = unbundle $ fetchBlock fetchInstr
        fetchInstr = instrBRAM addr (pure Nothing)
```

- testSystem

```
Nop
Nop
Push 2.0
Push 10.0
Calc Mult
Nop
Nop
Push 3.0
Push 4.0
Push 11.0
Calc Add
Nop
Nop
Calc Mult
Nop
Nop
Calc Add
Nop
Nop
Push 12.0
Push 5.0
Calc Add
Nop
Nop
Calc Mult
Nop
Nop
Nop
Nop
Nop
Nop
Nop
```

5 Exc5

```
system :: HiddenClockResetEnable dom => Signal dom (Maybe Proc.Value)
system = output
  where
    output = Proc.system procInstr
    procInstr = Fetch.system
```

- testSystem

```
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Just 20.0
Nothing
Nothing
Nothing
Nothing
Nothing
Just 15.0
Nothing
Nothing
Just 45.0
Nothing
Nothing
Just 65.0
Nothing
Nothing
Nothing
Nothing
Just 17.0
Nothing
Nothing
Just 127.99609375
Nothing
Nothing
Nothing
Nothing
Nothing
```

6 Exc6

```haskell
data ProcState
  = Calcing (Maybe Value) Opcode
  | Saving RegisterAddress
  | Idle
  deriving (Show, Generic, NFDataX)
```

```haskell
processor :: State -> (Instr, Value) -> (State, (Stack.Instr, Value))
processor (state, regs) (instr, value) = case state of
  Idle -> case instr of
    Push v      -> ((Idle, regs), (Stack.Push v, regs !! 0))
    PushR addr  -> ((Idle, regs), (Stack.Push $ regs
                                                  !! addr
                                                  , regs !! 0))
    Save addr   -> ((Saving addr, regs), (Stack.Pop, regs !! 0))
    Calc opCode -> ((Calcing Nothing opCode, regs)
                     , (Stack.Pop, regs !! 0))
    _           -> ((Idle, regs), (Stack.Nop, regs !! 0))
  Calcing storedValue opCode -> output
    where
      operation = case opCode of
        Add  -> (+)
        Mult -> (*)
      output = case storedValue of
        Just previous -> ((Idle, regs)
                           , (Stack.Push opResult, regs !! 0))
            where opResult = operation value previous
        Nothing -> ((Calcing (Just value) opCode, regs)
                     , (Stack.Pop, regs !! 0))
  Saving addr -> ((Idle, newRegs)
                   , (Stack.Nop, regs !! 0))
                     where newRegs = replace addr value regs
```

```haskell
procBlock :: HiddenClockResetEnable dom
  => Signal dom (Instr, Value) -> Signal dom (Stack.Instr, Value)
procBlock = mealy processor (Idle, replicate d4 5)
```

```haskell
system :: HiddenClockResetEnable dom
  => Signal dom Instr -> Signal dom Value
system instr = output
  where
    readResult = Stack.system stackInstruction
    (stackInstruction, output) = unbundle
                                  $ procBlock
                                  $ bundle (instr, readResult)
```

```
system' :: HiddenClockResetEnable dom
  => RegisterFile -> Signal dom Instr -> Signal dom Value
system' regs instr = output
  where
    procBlock' = mealy processor (Idle, regs)
    readResult = Stack.system stackInstruction
    (stackInstruction, output) = unbundle
                                   $ procBlock'
                                   $ bundle (instr, readResult)
```

- testSystem:

```
[
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    5,
    18
]
```

- testSystem'

```
[
    1,
    1,
    1,
    1,
    1,
    1,
    1,
    1,
    1,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    3,
    18
]
```

7 Exc7

```haskell
fetcher :: State -> Proc.Instr -> (State, Output)
fetcher (pc, stallt) instr
    | stallt > 0                = ((pc, stallt - 1), (Proc.Nop, pc))
    | otherwise                 = ((pc + 1, delay), (instr, pc + 1))
      where
          delay = case instr of
              Proc.Calc _ -> 2
              Proc.Save _ -> 1
              _           -> 0
```

```
instrBRAM :: HiddenClockResetEnable dom
  => Signal dom Address -> Signal dom (Maybe (Address, Proc.Instr))
                                                  -> Signal dom Proc.Instr
instrBRAM = blockRam $ testProgram
```

```
system :: HiddenClockResetEnable dom => Signal dom Proc.Instr
system = procInstr
    where
        (procInstr, addr) = unbundle $ fetchBlock fetchInstr
        fetchInstr = instrBRAM addr (pure Nothing)
```

- testSystem

```
Nop
Nop
PushR 2
Push 0.9765625
Calc Mult
Nop
Nop
Save 2
Nop
PushR 2
Push 0.0390625
Calc Mult
Nop
Nop
PushR 1
Calc Add
Nop
Nop
Save 1
Nop
PushR 1
Push 0.0390625
Calc Mult
Nop
Nop
PushR 0
Calc Add
Nop
Nop
Save 0
Nop
PushR 2
Push 0.9765625
Calc Mult
Nop
Nop
Save 2
Nop
PushR 2
Push 0.0390625
```

8 Exc8

8.1

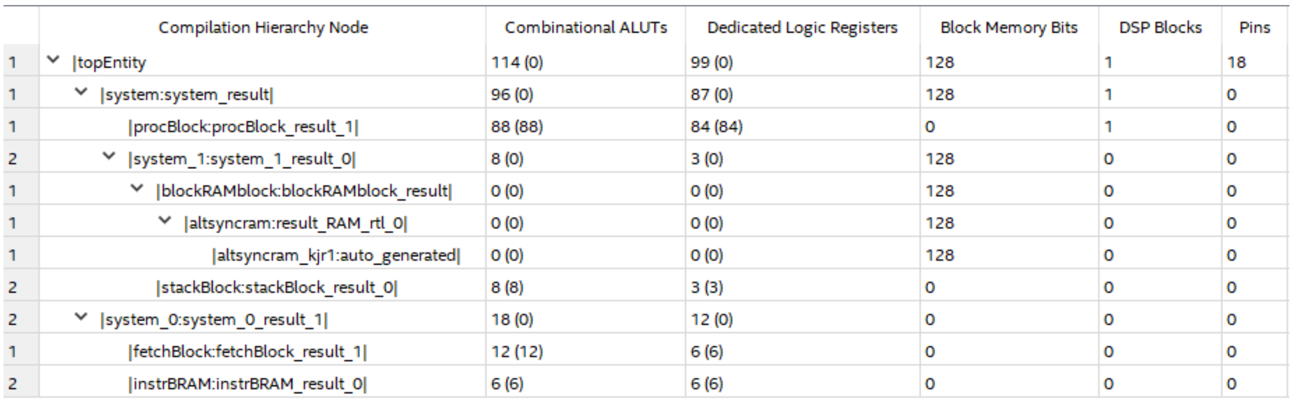
```
system :: HiddenClockResetEnable dom => Signal dom (Proc.Value)
system = output
  where
    output = Proc.system procInstr
    procInstr = Fetch.system
```

```
system' :: HiddenClockResetEnable dom
    => Vec 4 (Proc.RegisterFile)
        -> Signal dom (Vec 4 (Proc.Value))
system' regs = outputSignals
  where
    instrStream = Fetch.system

    proc0 = Proc.system' (regs !! 0) instrStream
    proc1 = Proc.system' (regs !! 1) instrStream
    proc2 = Proc.system' (regs !! 2) instrStream
    proc3 = Proc.system' (regs !! 3) instrStream

    outputSignals = bundle (proc0 :> proc1 :> proc2 :> proc3 :> Nil)
```

- testSystem

```
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.0
5.19921875
5.19921875
5.19921875
5.19921875
5.19921875
5.19921875
5.19921875
5.19921875
5.19921875
```

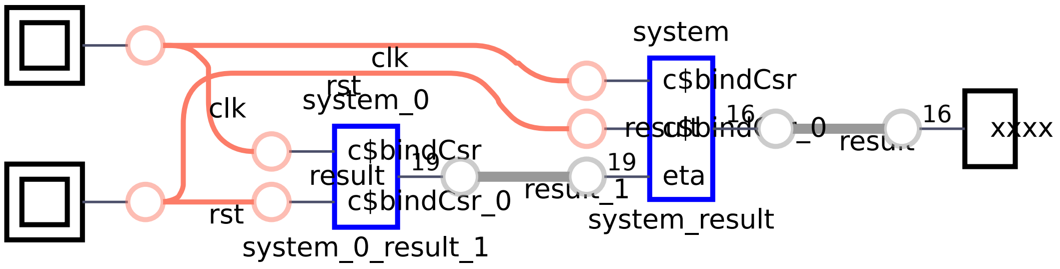
- Register and DSP Count:



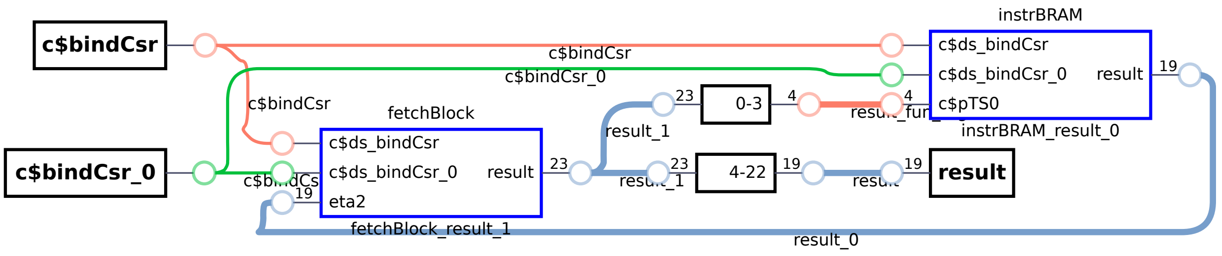
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins
1	|topEntity	114 (0)	99 (0)	128	1	18
1	|system:system_result|	96 (0)	87 (0)	128	1	0
1	|procBlock:procBlock_result_1|	88 (88)	84 (84)	0	1	0
2	|system_1:system_1_result_0|	8 (0)	3 (0)	128	0	0
1	|blockRAMblock:blockRAMblock_result|	0 (0)	0 (0)	128	0	0
1	|altsyncram:result_RAM_rtl_0|	0 (0)	0 (0)	128	0	0
1	|altsyncram_kjr1:auto_generated|	0 (0)	0 (0)	128	0	0
2	|stackBlock:stackBlock_result_0|	8 (8)	3 (3)	0	0	0
2	|system_0:system_0_result_1|	18 (0)	12 (0)	0	0	0
1	|fetchBlock:fetchBlock_result_1|	12 (12)	6 (6)	0	0	0
2	|instrBRAM:instrBRAM_result_0|	6 (6)	6 (6)	0	0	0

(a) topEntity Summary

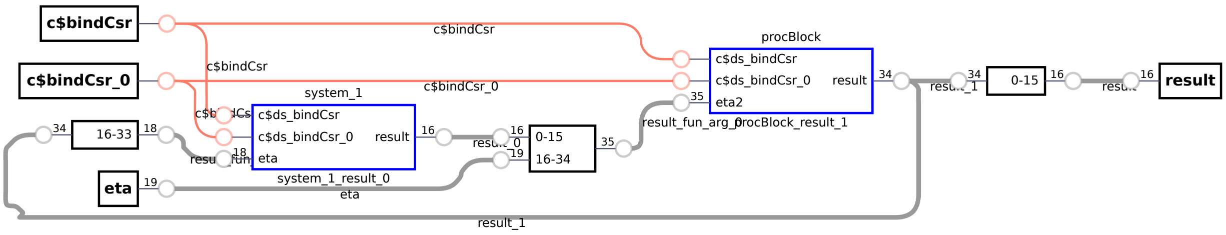
After generating the Verilog and loading and synthesis in Quartus, it is reported that there is 1 DSP and 99 Registers. This makes sense since our DSP is found in the procBlock, hanlding the execution whilst our blockRAM handles data.



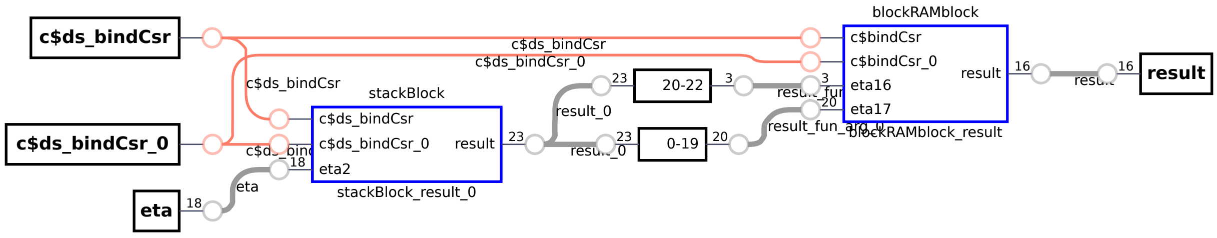
(b) Top Level



(c) Fetch system - "system_0"

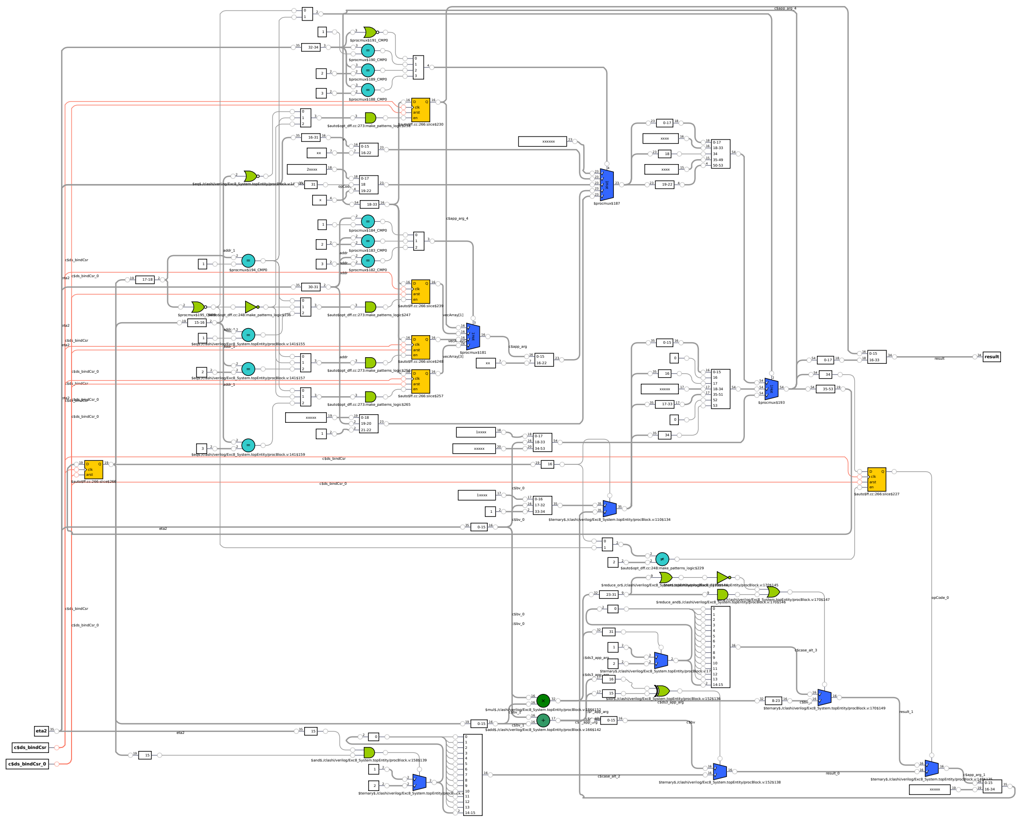


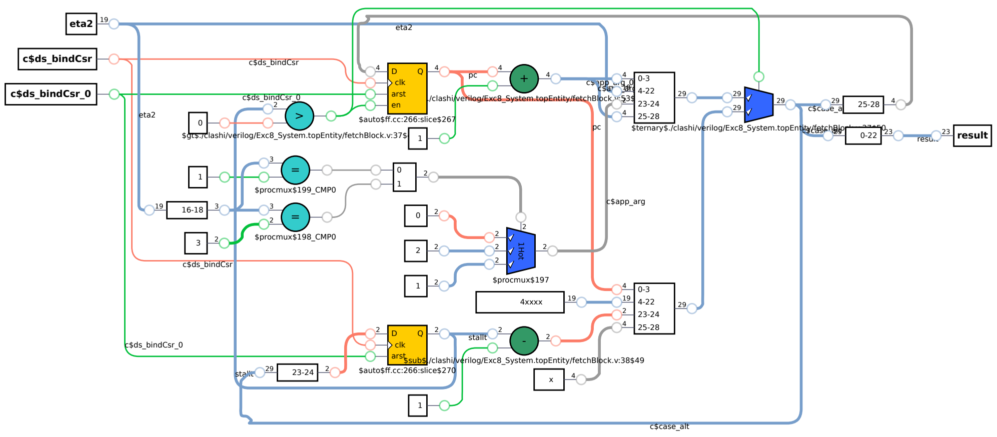
(d) Processing system - "system"



(e) Stack system - "system_1"

Figure 1: System connections


(a) Processor block


(b) Fetch block

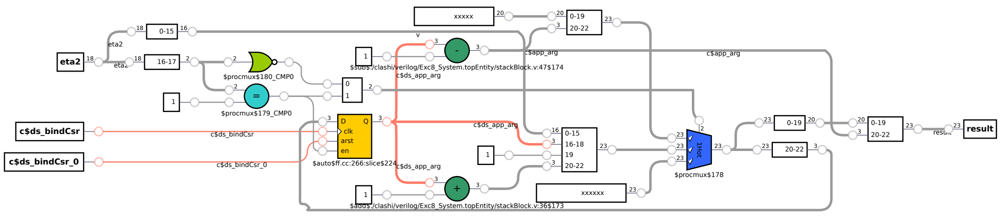

(c) Stack block

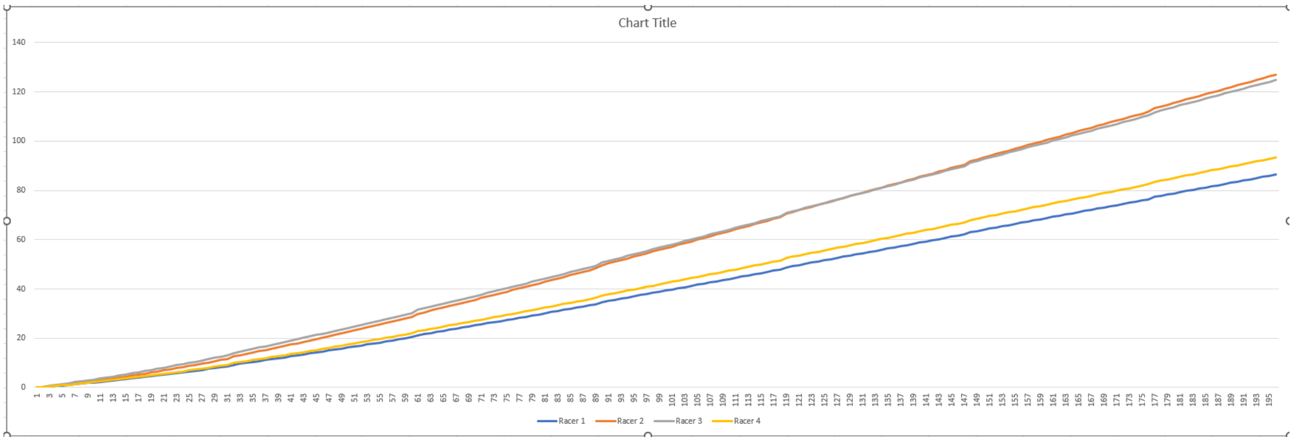
Figure 2: Developed Blocks

8.2

```haskell
-- Function to modify regs with base and offset scale
regsMod :: Unsigned 2 -> Proc.RegisterFile
regsMod i =
  let base = 5.0
      offset = fromIntegral i * 0.5
  in 0.0
  :> (base + offset * 0.2)
  :> (base + offset * 0.3)
  :> 0.0 :> Nil

-- Combines usr regs with modified
regsAlt :: Unsigned 2 -> Proc.RegisterFile -> Proc.RegisterFile
regsAlt i regs = zipWith (+) regs (regsMod i)

system' :: HiddenClockResetEnable dom
    => Vec 4 (Proc.RegisterFile) -> Signal dom (Vec 4 (Proc.Value))
system' regs = outputSignals
  where
    instrStream = Fetch.system
    regsNew = zipWith regsAlt (0 :> 1 :> 2 :> 3 :> Nil) regs
    procs = map (\regsNew -> Proc.system' regsNew instrStream) regsNew
    outputSignals = bundle procs
```

- clashi - inputs

```
clashi> let regs0 = 0:>0:>0:>0:>Nil
clashi> let regs1 = 0:>1:>3:>0:>Nil
clashi> let regs2 = 0:>3:>1:>0:>Nil
clashi> let regs3 = 0:>0:>0:>0:>Nil
clashi> let allRegs = regs0 :> regs1 :> regs2 :> regs3 :> Nil
clashi> wf allRegs
```



(a) Race results for above input