SOC Design
Steps to Construct a Design

Jiin Lai

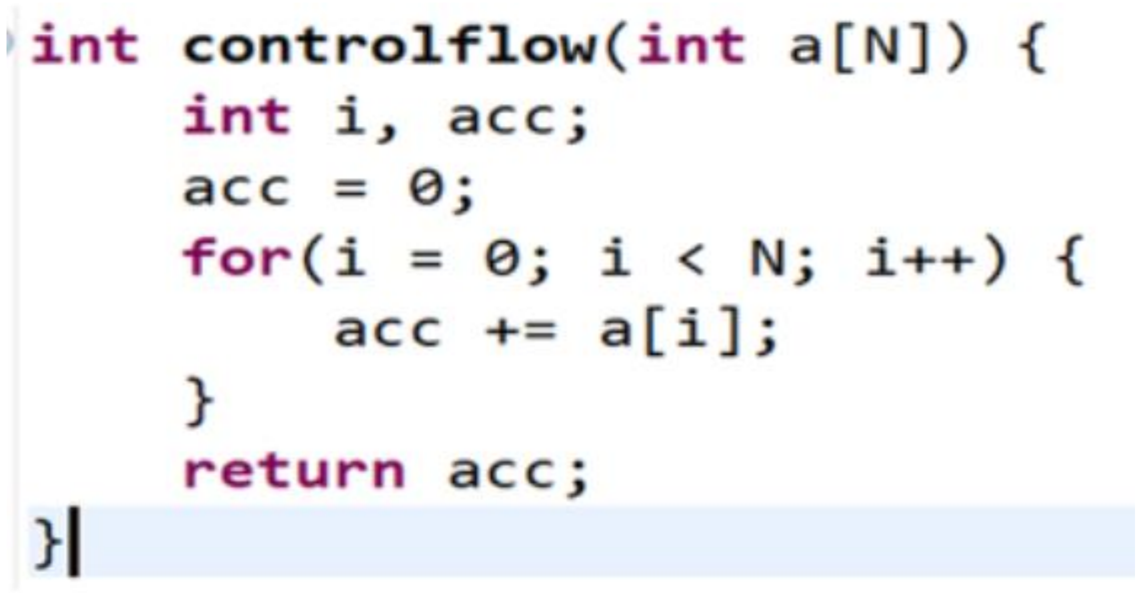
Flow to Construct a Design (reference: verilog-hls)

Steps

1. Given a functional specification, specified by high-level language, e.g. C-model
2. With target PPA (Performance/Power/Area), define its datapath
    1. Resource allocation & Scheduling
    2. Identify datapath components & its control signals, e.g. mux select, register enable, operator
3. Draw Timing Diagram to exercise the function,
    1. Refine datapath
    2. Help FSM design
4. Decompose into States considering timing & resource sharing
5. Design the controller (FSM), and generate control signals

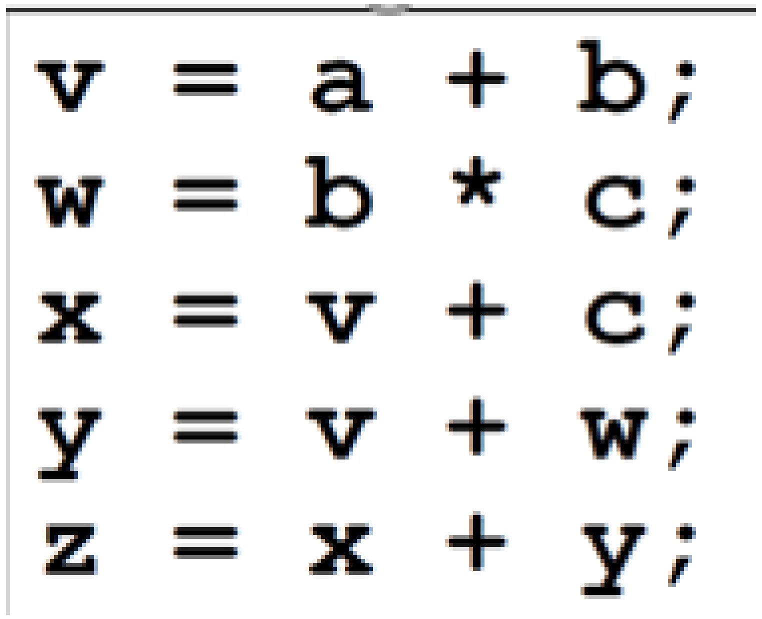
Step 1: Function Specification by C-model

Loop Control



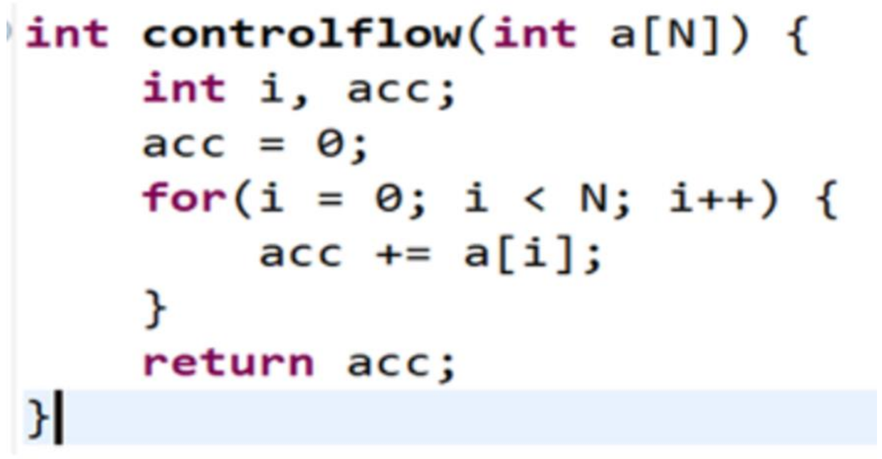
```
int controlflow(int a[N]) {
    int i, acc;
    acc = 0;
    for(i = 0; i < N; i++) {
        acc += a[i];
    }
    return acc;
}
```

Expression

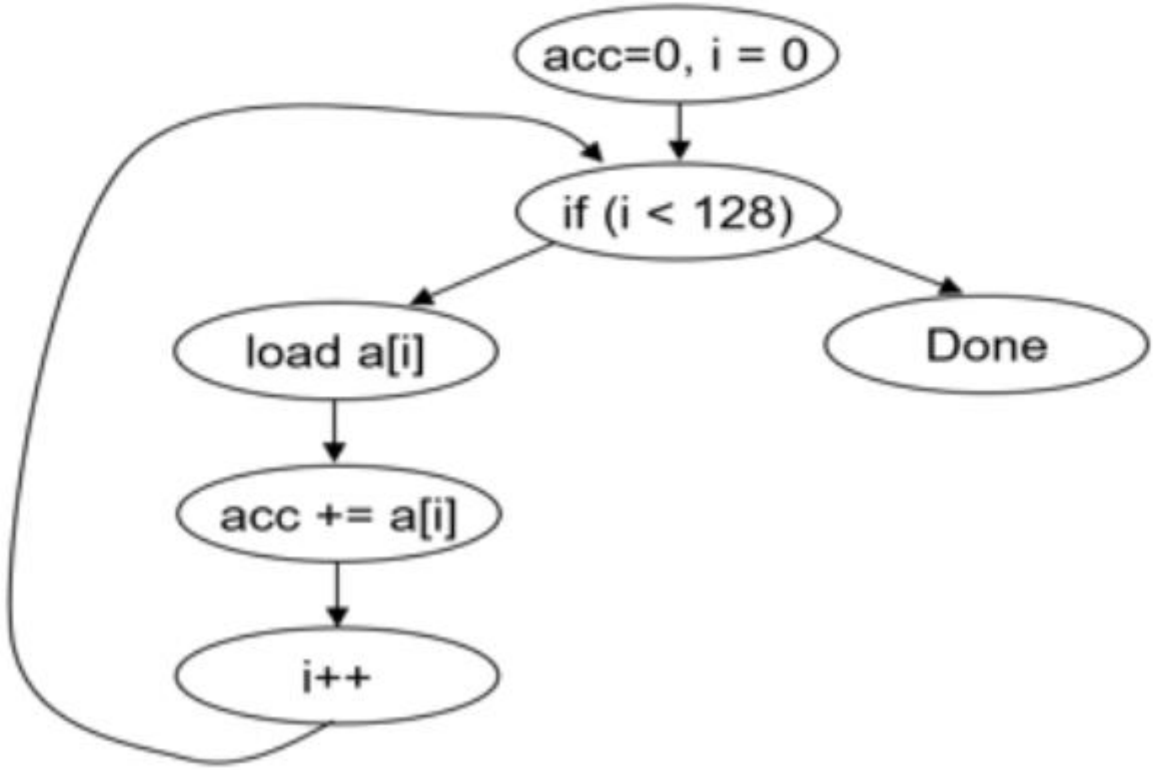


```
v = a + b;
w = b * c;
x = v + c;
y = v + w;
z = x + y;
```

Step 2-1 : Draw Dataflow

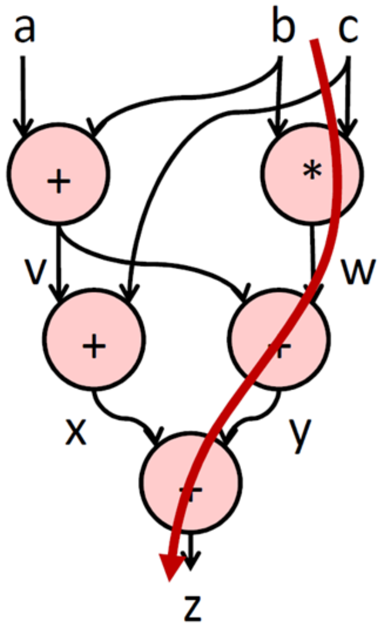


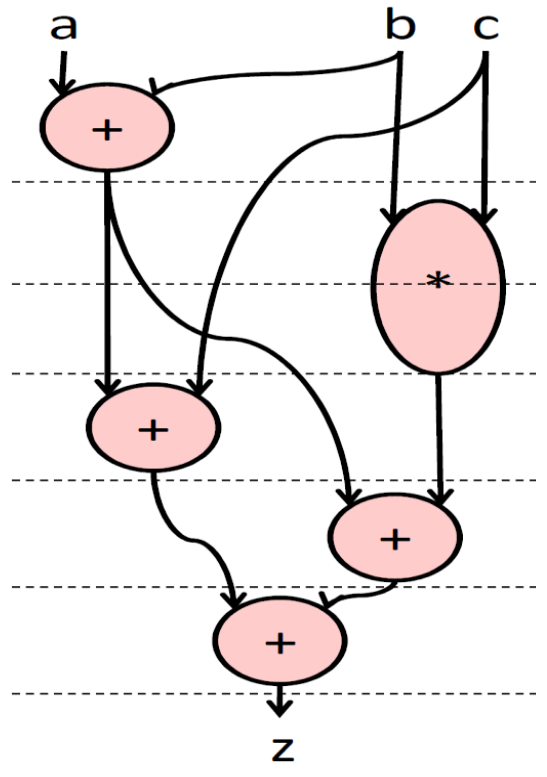
```
int controlflow(int a[N]) {
    int i, acc;
    acc = 0;
    for(i = 0; i < N; i++) {
        acc += a[i];
    }
    return acc;
}
```


$$\begin{tikzcd}
 & \text{acc=0, i = 0} \arrow[d] & \\
 & \text{if (i < 128)} \arrow[dl] \arrow[dr] & \\
\text{load a[i]} \arrow[d] & & \text{Done} \\
\text{acc += a[i]} \arrow[d] & & \\
\text{i++} \arrow[uuur, bend left=60] & &
\end{tikzcd}$$

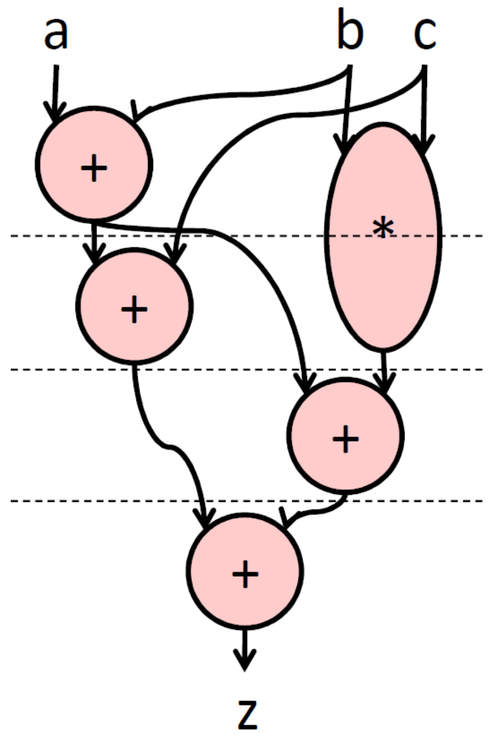
Step 2 - 1: Draw dataflow (DAG) Resource allocation & Scheduling

```
v = a + b;
w = b * c;
x = v + c;
y = v + w;
z = x + y;
```



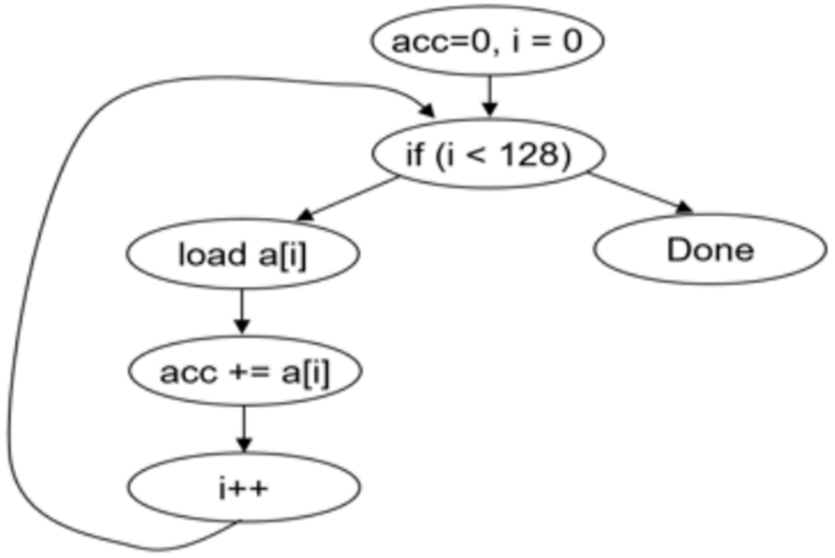


delay=6 using 1 MAC

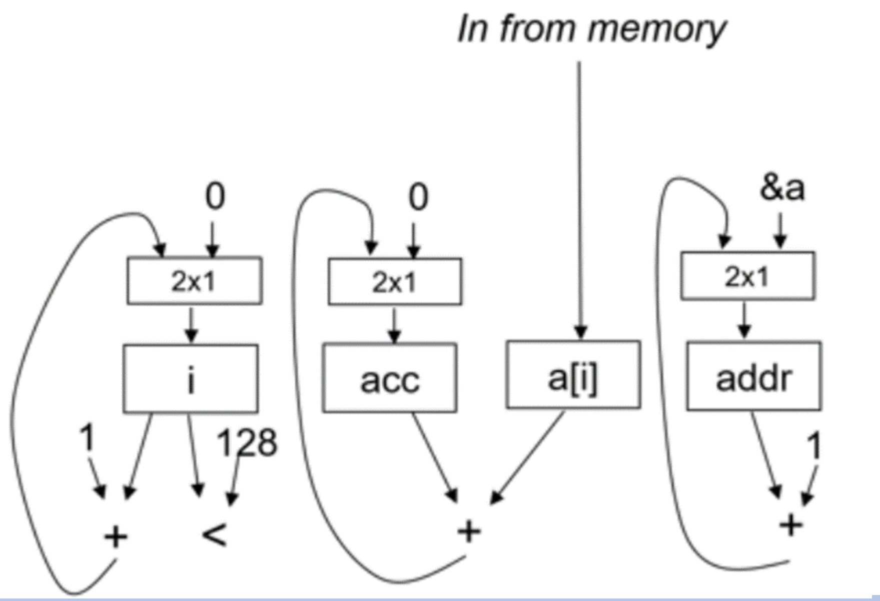


delay=4 using 1 adder and 1 multiplier

Step 2-2 Design Datapath



- Datapath components
  - Flip-flop (storage) – state variables
  - Multiplexer
  - Operator
- Resource Scheduling and Sharing

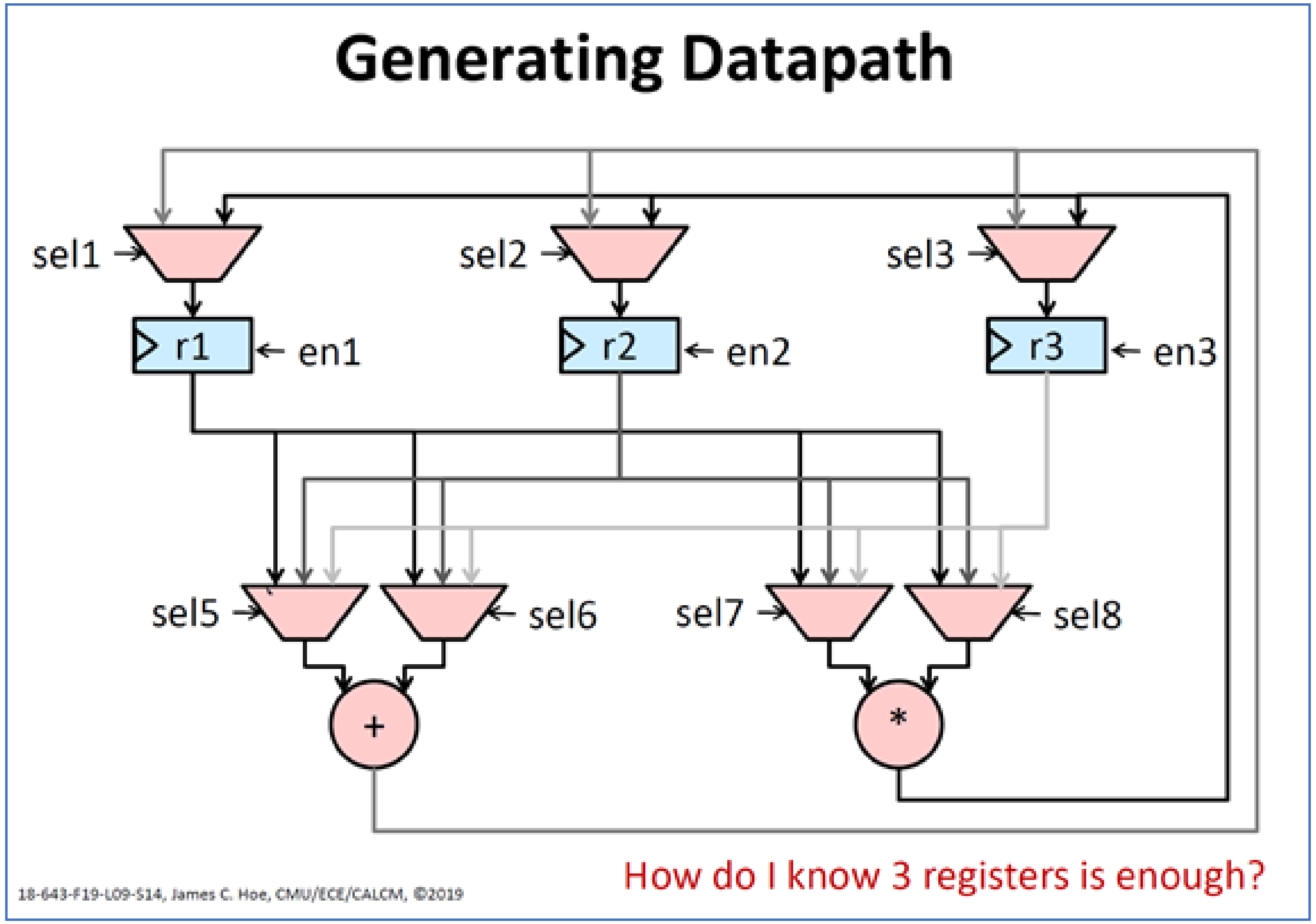


Multiplexer

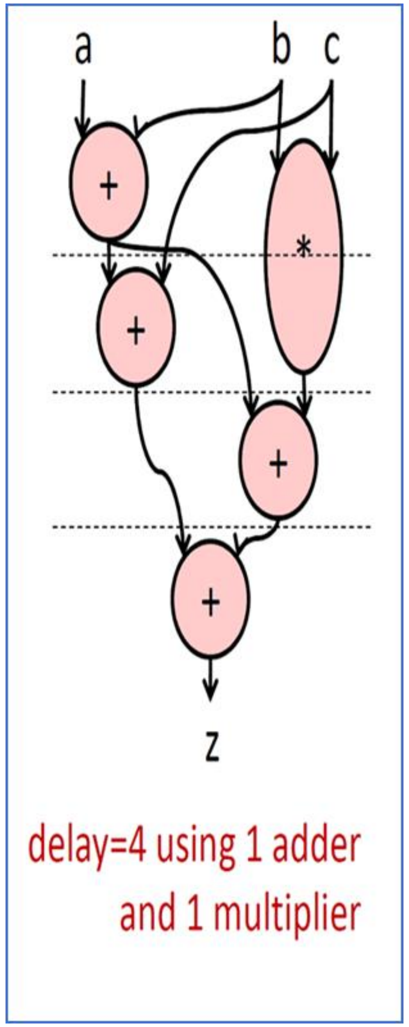
State Variables

Operators

Step 2-2: Design Datapath - Binding & Resource Sharing



How do I know 3 registers is enough?

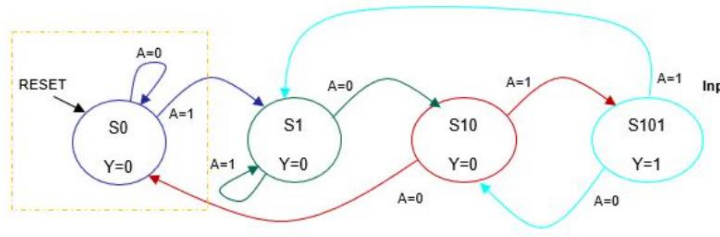


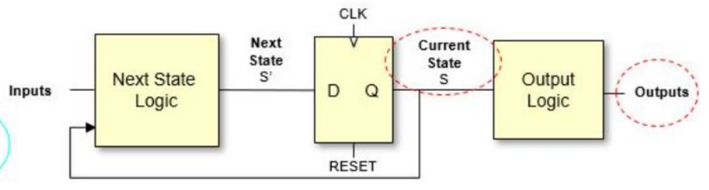
delay=4 using 1 adder and 1 multiplier

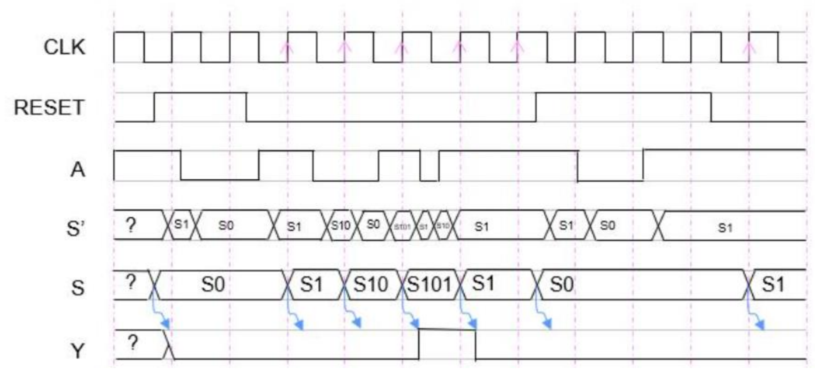
Step3: Draw Timing Diagram

A closer look at the design by considering

- Input / output Interface timing/protocol
- Timing estimation, e.g. a multiplier takes multiple cycle
- Introduce handshake signals, e.g. when multicycle multiplication is done?

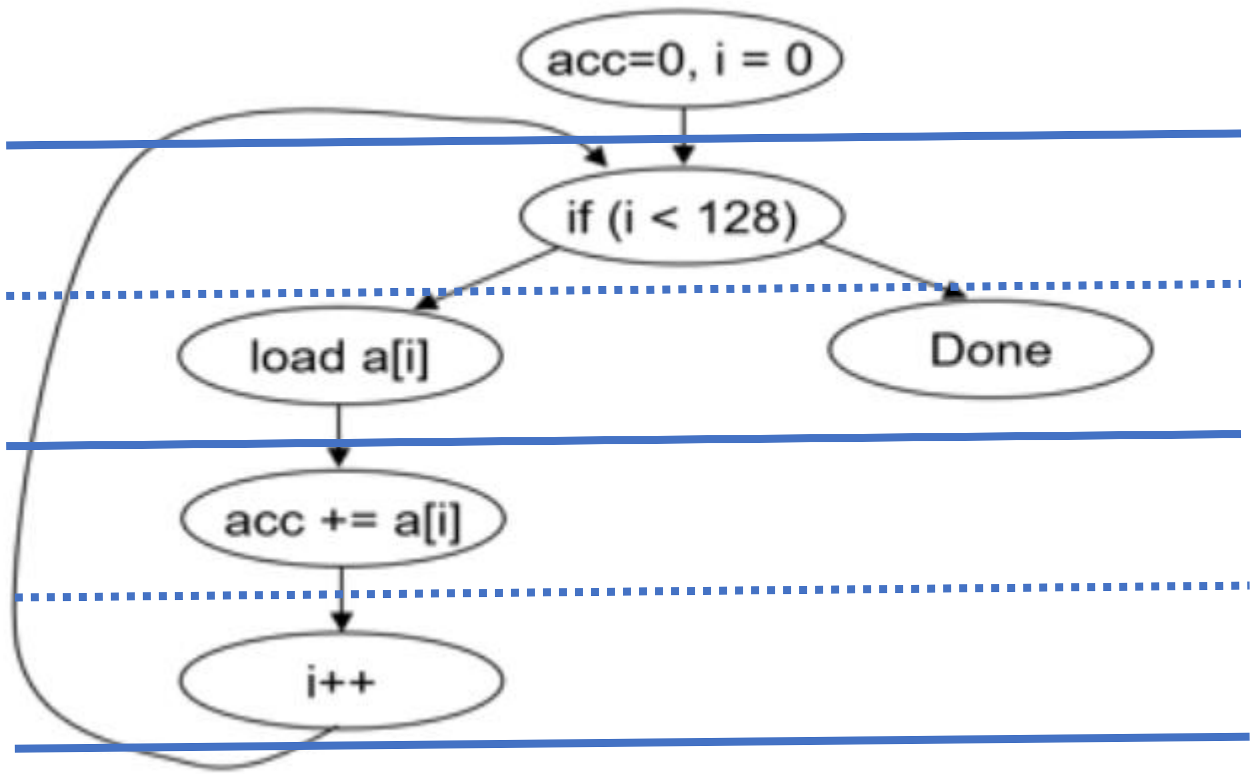


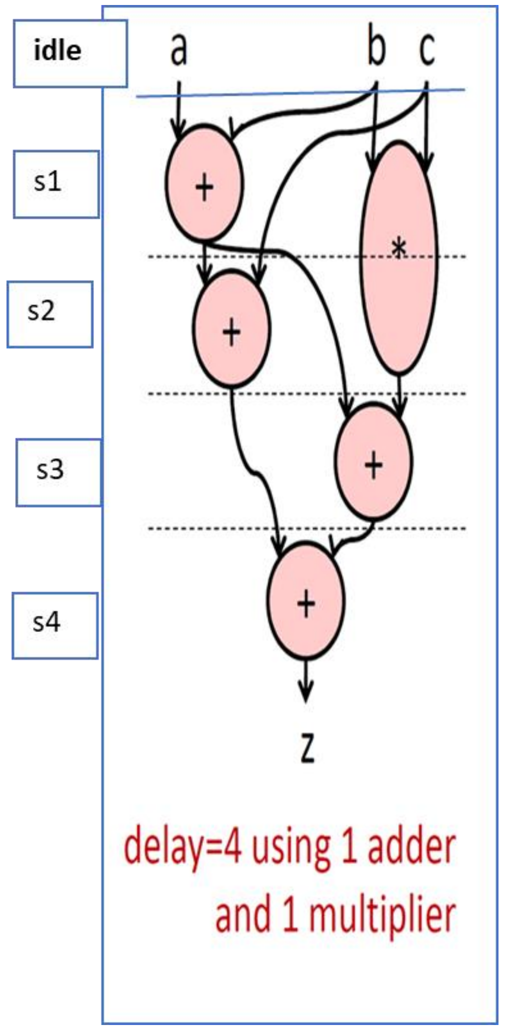




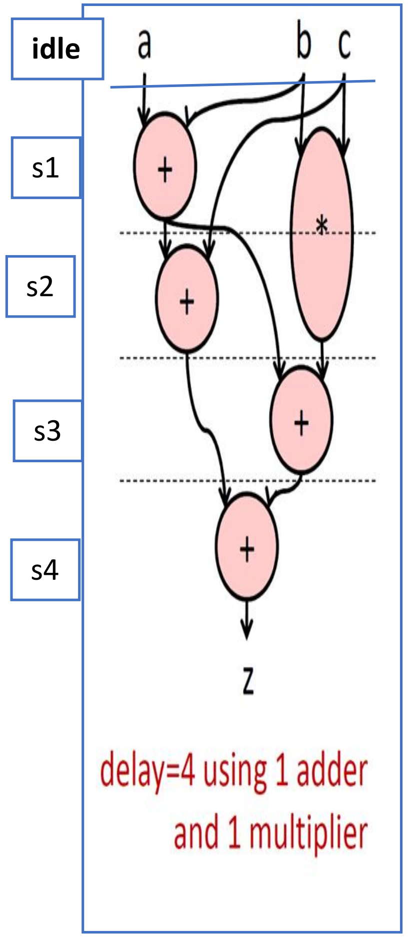
https://gacaffe.net/en/2019/04/30/about-timing-diagrams-of-moore-finite-state-machines/

Step 4: Decompose into States





Step 5 – Design the Controller and Generate control Signals



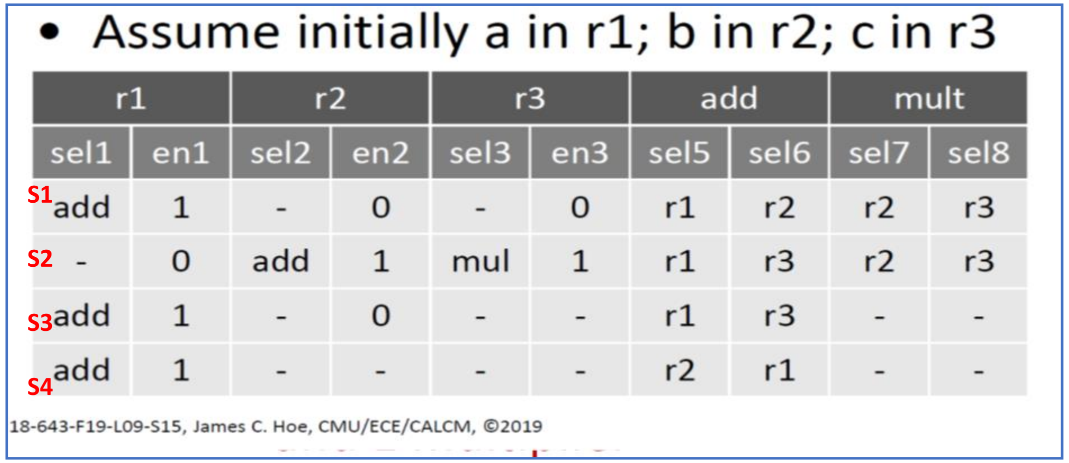
Generating State Machines

- Multiple cycles
- States: (idle, s1, s2, s3, s4
- Encode States: 000,100,101,110,111)

Generate Control Signals

- Combines the Control Signal State & State Machine, e.g.
- e.g. en1 = s1 | s3 | s4;

Assign State for Control Signals

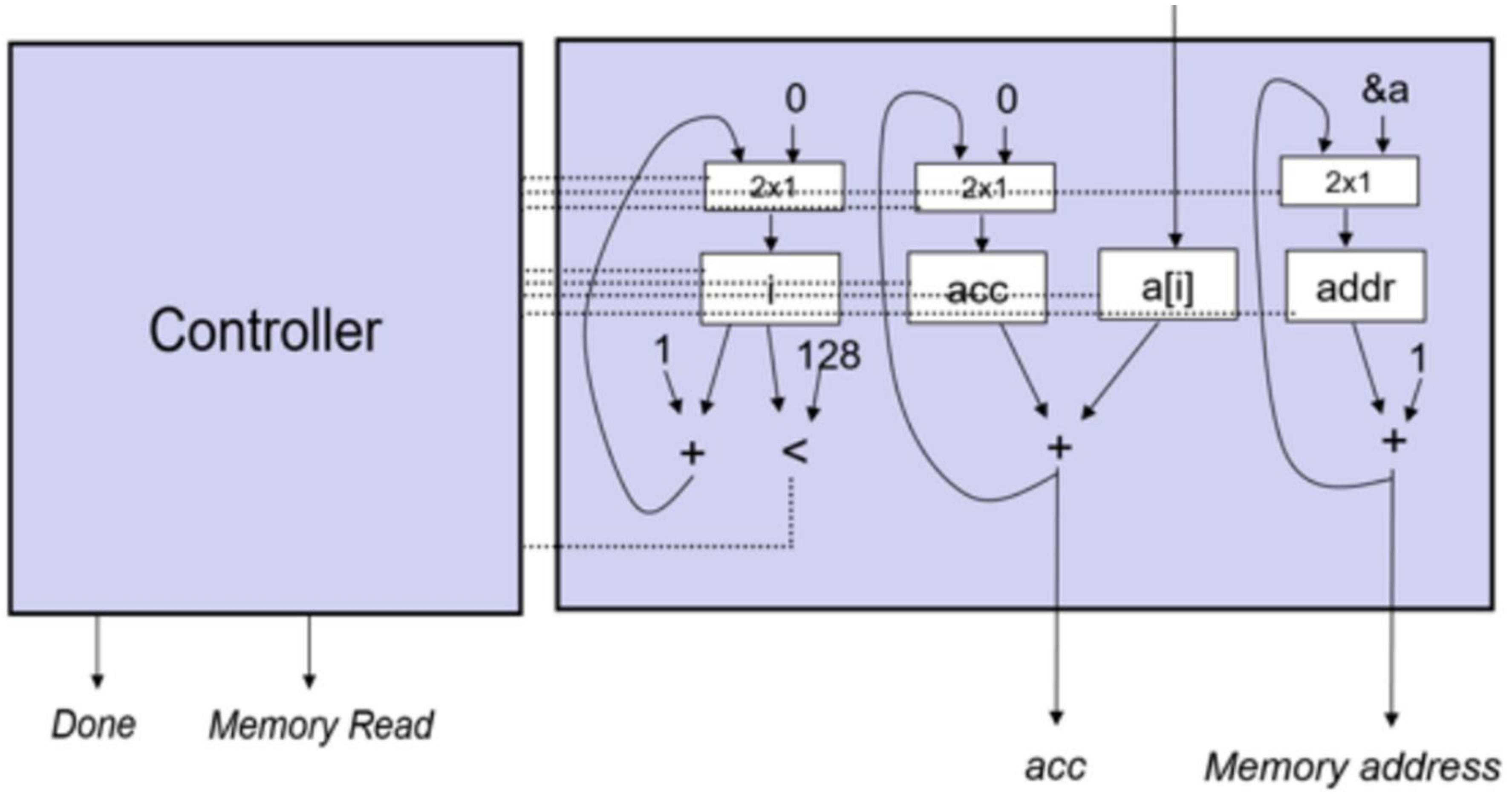


- Assume initially a in r1; b in r2; c in r3

	r1		r2		r3		add		mult	
	sel1	en1	sel2	en2	sel3	en3	sel5	sel6	sel7	sel8
S1	add	1	-	0	-	0	r1	r2	r2	r3
S2	-	0	add	1	mul	1	r1	r3	r2	r3
S3	add	1	-	0	-	-	r1	r3	-	-
S4	add	1	-	-	-	-	r2	r1	-	-

18-643-F19-L09-S15, James C. Hoe, CMU/ECE/CALCM, ©2019

Compose Datapath and Controller



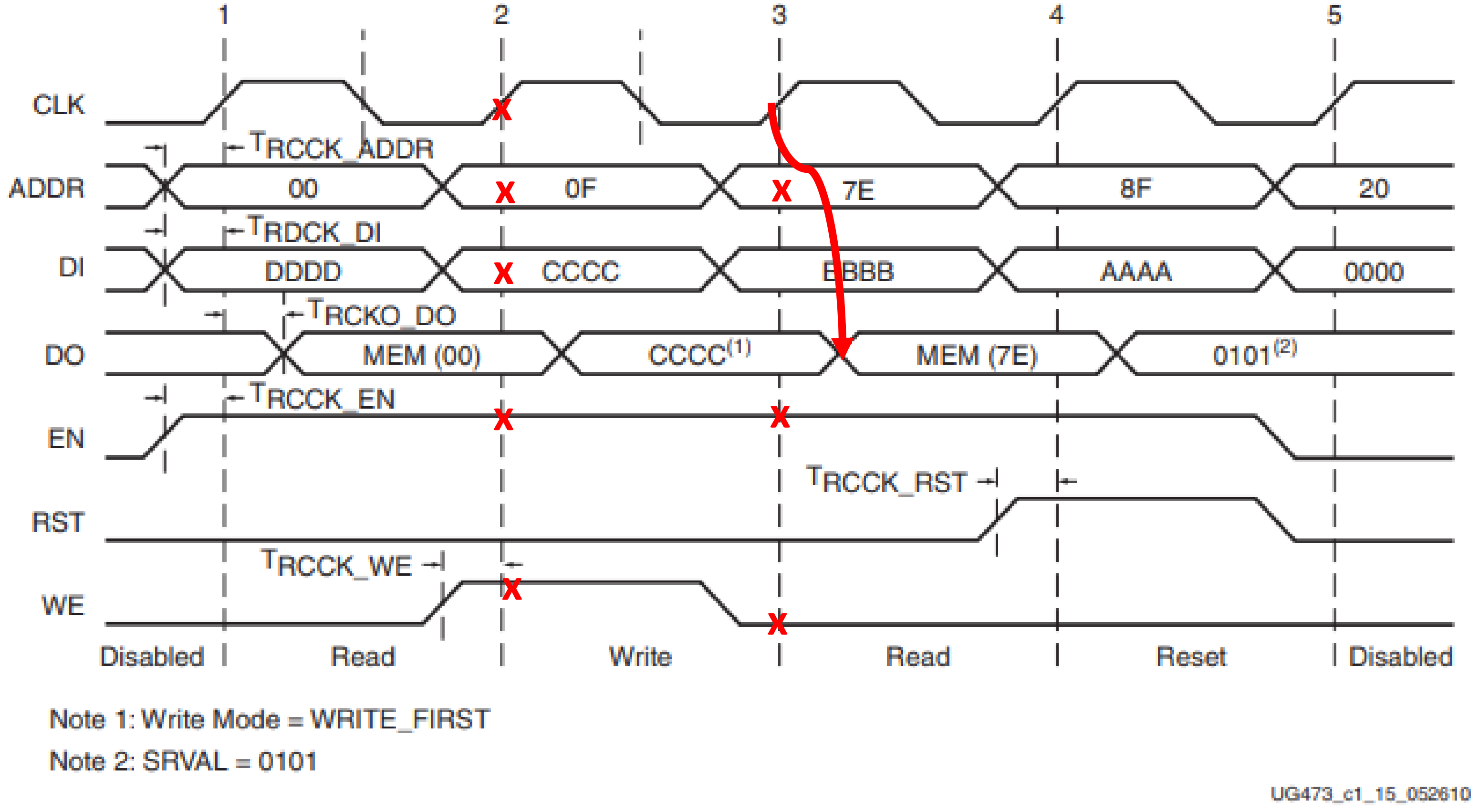
An AXI Stream Design Example

Design an AXI Stream master to achieve back-to-back zero-state data transfer. The data is sourced from SRAM which has 1 clock delay synchronous read.
Illustrate how timing waveform assists controller design

SRAM Access Timing (Synchronous Read)

Write data is written when WE/EN is sampled

Read data is available in next cycle, when EN is sampled

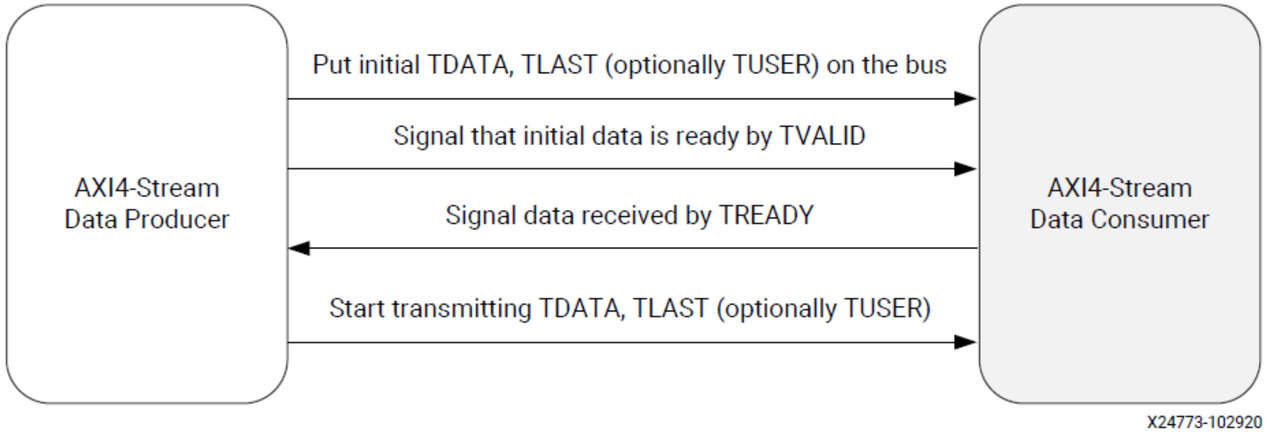


Note 1: Write Mode = WRITE_FIRST

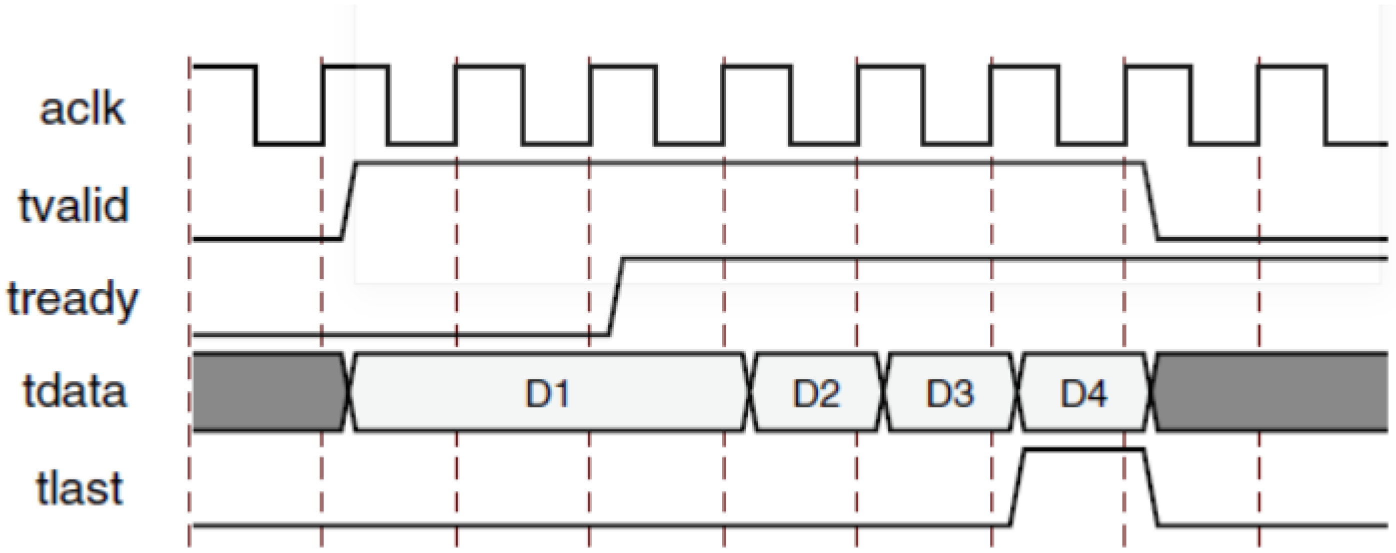
Note 2: SRVAL = 0101

SRAM access has different modes, refer to https://docs.xilinx.com/r/en-US/am007-versal-memory/Read-Operation?tocId=VRYu0HURA1U147fufYDMNQ

AXI4-Stream Transfer Protocol

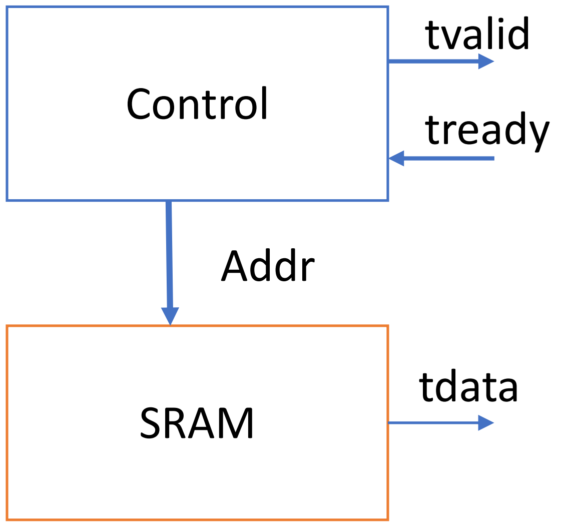


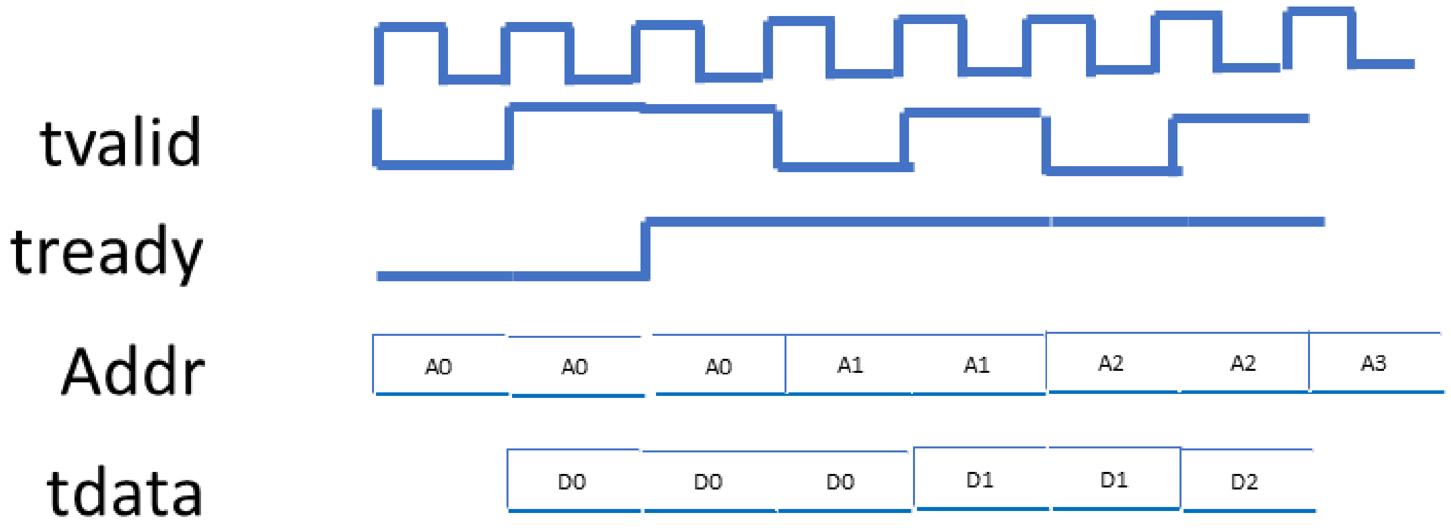
AXI4-Stream Data Producer
Put initial TDATA, TLAST (optionally TUSER) on the bus
Signal that initial data is ready by TVALID
Signal data received by TREADY
Start transmitting TDATA, TLAST (optionally TUSER)
AXI4-Stream Data Consumer
X24773-102920



Initial Design

SRAM directly supplies data to axi-stream data bus. Due to the 1T synchronous read data delay, the data transfer rate is **2-2-2** even with tready asserts





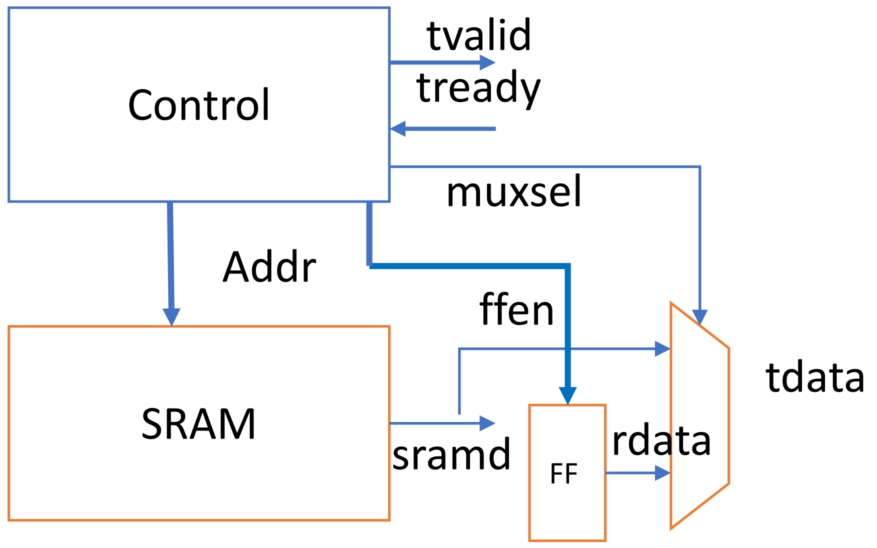
How to achieve 1-1-1 transfer rate with synchronous SRAM

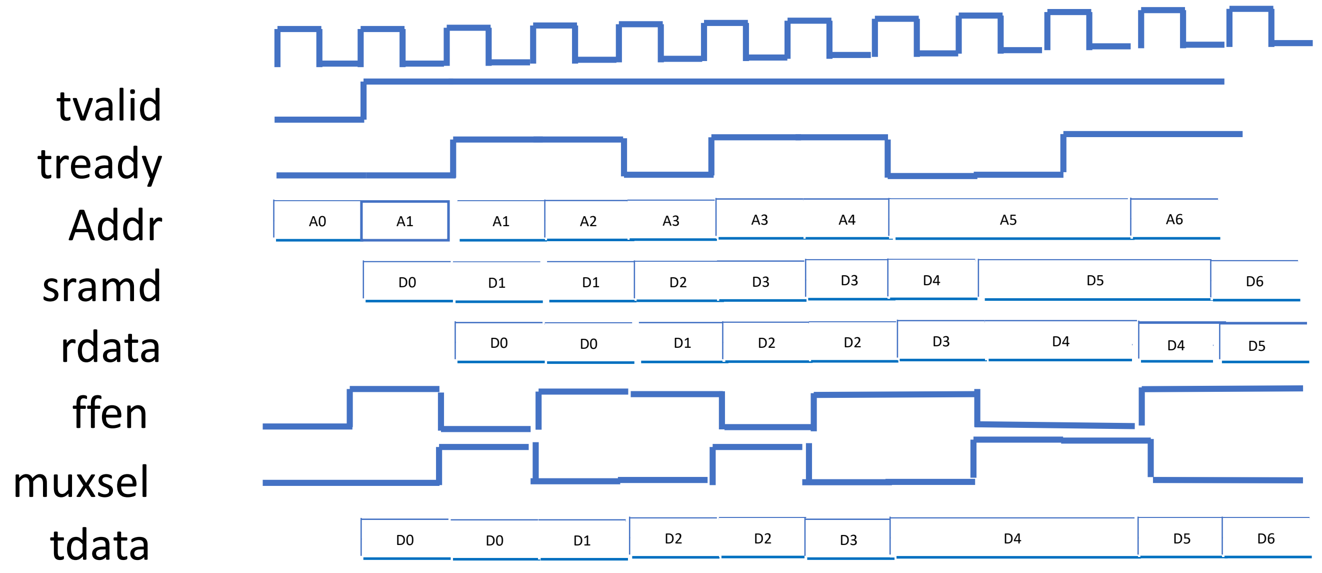
Pipeline Design to achieve 1-1-1 back-back data transfer

Since the SRAM has 1T read delay, intuitively,

1. we will need to prefetch the data, i.e. Advance Address at start, then, advance when tready sampled
2. Need to keep the SRAM data in case tready is not asserted. (FF)
3. A muxsel to choose tdata from SRAM output (sdramd) or latched data (rdata)

Use the timing waveform to assist the design process

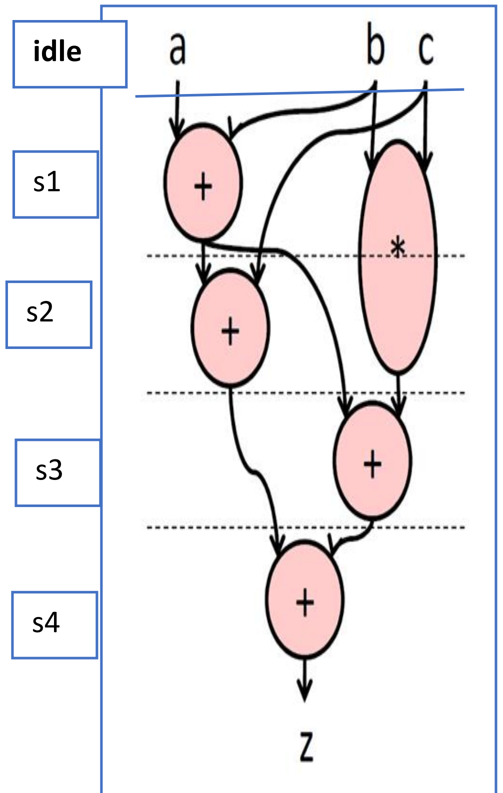




tvalid
tready
Addr
A0 A1 A1 A2 A3 A3 A4 A5 A6
sramd
D0 D1 D1 D2 D3 D3 D4 D5 D6
rdata
D0 D0 D1 D2 D2 D3 D4 D4 D5
ffen
muxsel
tdata
D0 D0 D1 D2 D2 D3 D4 D5 D6

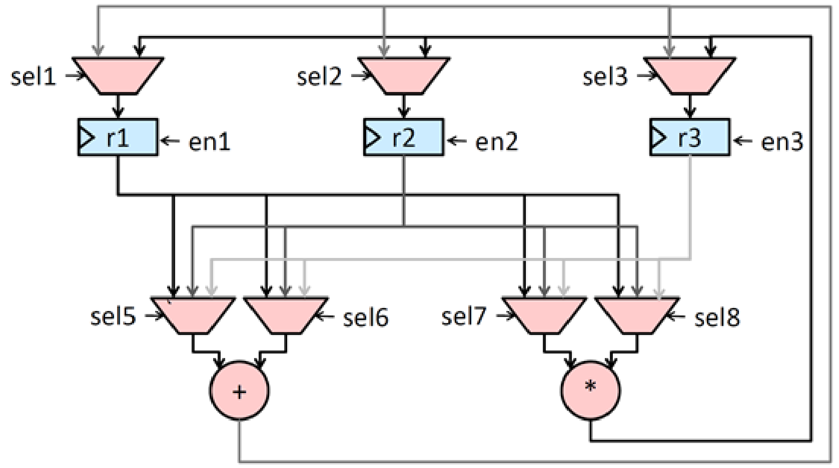
Supplement

Example – Expression Datapath Binding & Resource Sharing



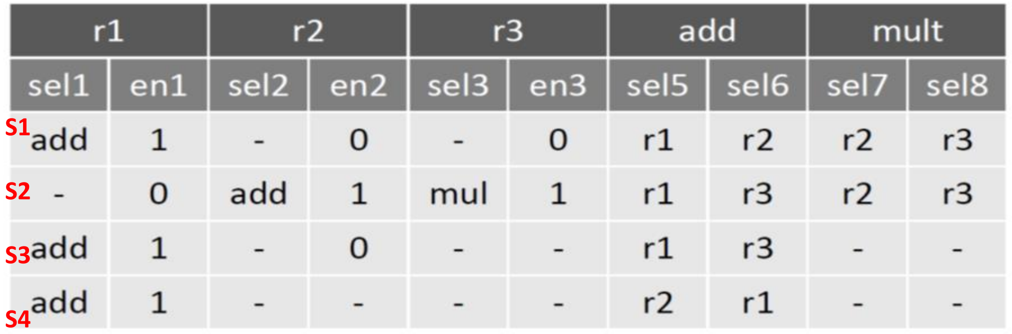
delay=4 using 1 adder and 1 multiplier

Generating Datapath



Assign State for Control Signals

- Assume initially a in r1; b in r2; c in r3

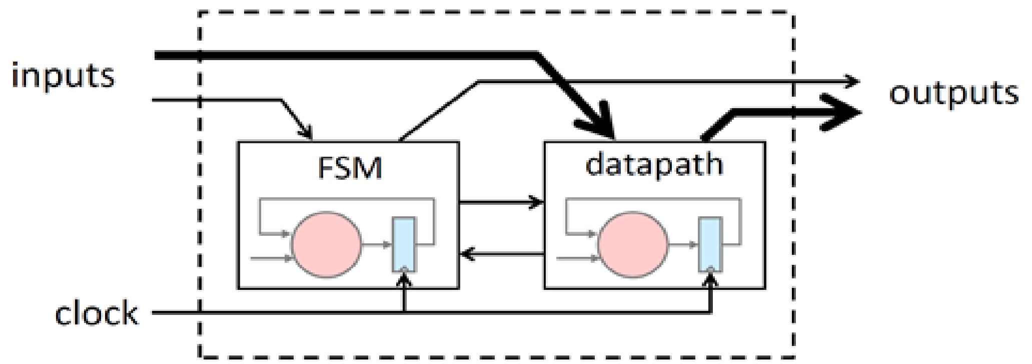


	r1		r2		r3		add		mult	
	sel1	en1	sel2	en2	sel3	en3	sel5	sel6	sel7	sel8
S1	add	1	-	0	-	0	r1	r2	r2	r3
S2	-	0	add	1	mul	1	r1	r3	r2	r3
S3	add	1	-	0	-	-	r1	r3	-	-
S4	add	1	-	-	-	-	r2	r1	-	-

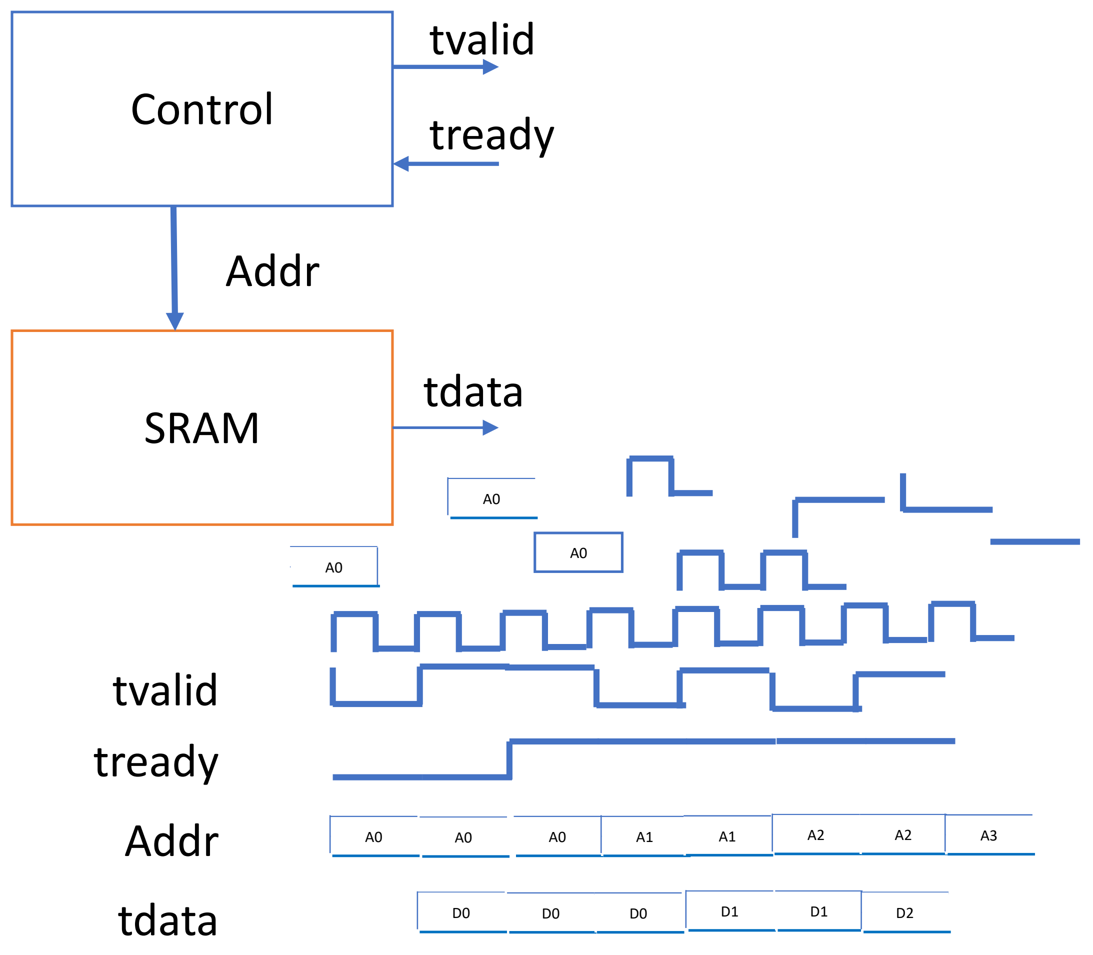
18-643-F19-L09-S15, James C. Hoe, CMU/ECE/CALCM, ©2019

Generating State Machines
- Multiple cycles
- States: (idle, s1, s2, s3, s4
- Encode States: 000,100,101,110,111)

Generate Control Signals
- Combines the Control Signal State & State Machine, e.g.
- e.g. en1 = s1 | s3 | s4;



Initial Design


Control
tvalid
tready
Addr
SRAM
tdata
A0
A0
A0
tvalid
tready
Addr
A0
A0
A0
A1
A1
A2
A2
A3
tdata
D0
D0
D0
D1
D1
D2