A look inside behavioral synthesis

Michael Meredith

Delft University of Technology

What is Synthesis

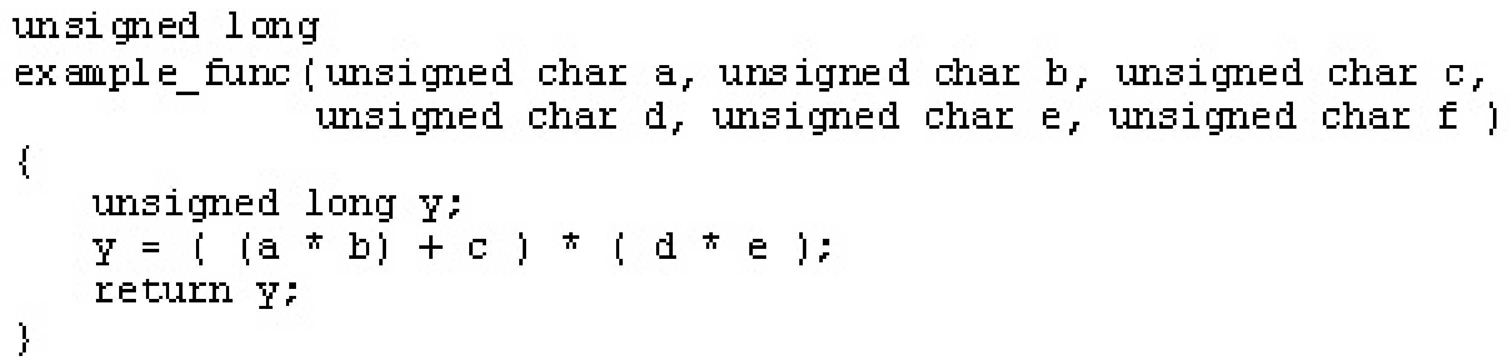
- Behavioral synthesis is an automated design process that interprets an algorithmic description of a desired behavior and creates hardware that implements that behavior.
- Starting with an algorithmic description in a high-level language, behavioral synthesis tools automatically create the cycle-by-cycle detail needed for hardware implementation. Most behavioral synthesis approaches leverage the existing logic synthesis toolset by creating a register transfer level (RTL) implementation from the algorithmic description.

Stages of the behavioral synthesis process

- *Lexical processing*
- *Algorithm optimization*
- *Control/Dataflow analysis*
- *Library processing*
- *Resource allocation*
- *Scheduling*
- *Functional unit binding*
- *Register binding*

Example

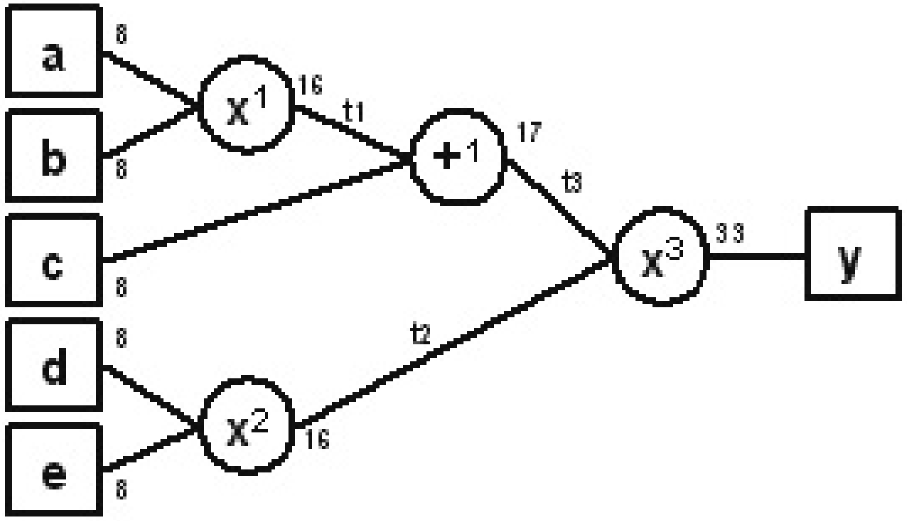
- Suppose we need to compute ((a * b) + c) * (d * e) where each value is 8 bits. We can express this in the C language as:



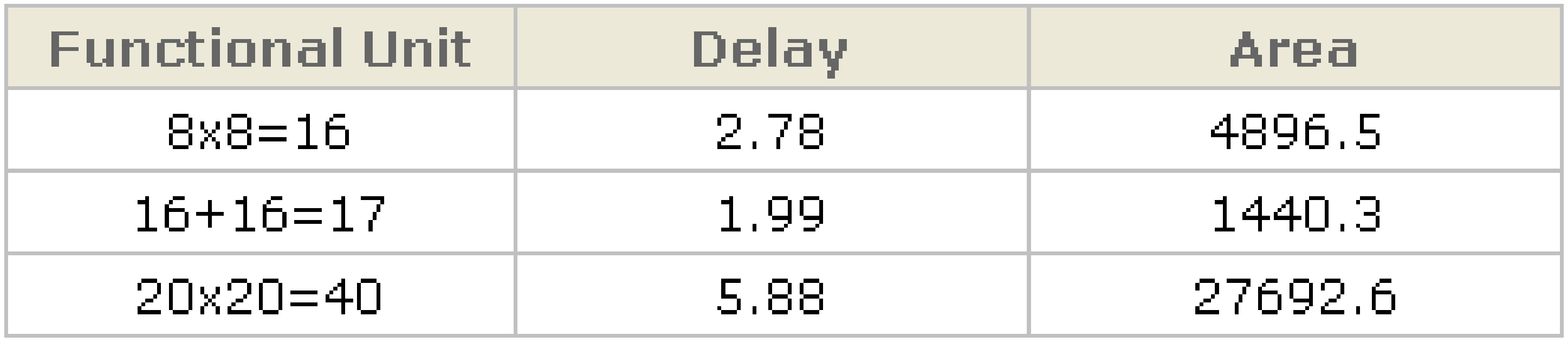
```
unsigned long
example_func(unsigned char a, unsigned char b, unsigned char c,
             unsigned char d, unsigned char e, unsigned char f )
{
    unsigned long y;
    y = ( (a * b) + c ) * ( d * e );
    return y;
}
```

- In order to build hardware to perform this computation, we will need to deal with I/O protocol, but it is useful to consider the algorithm in isolation to understand the behavioral synthesis process.

- The control/dataflow analysis would construct a graph from this code as follows:



- Assume our library contains the following functional units:



Functional Unit	Delay	Area
8x8=16	2.78	4896.5
16+16=17	1.99	1440.3
20x20=40	5.88	27692.6

TU Delft

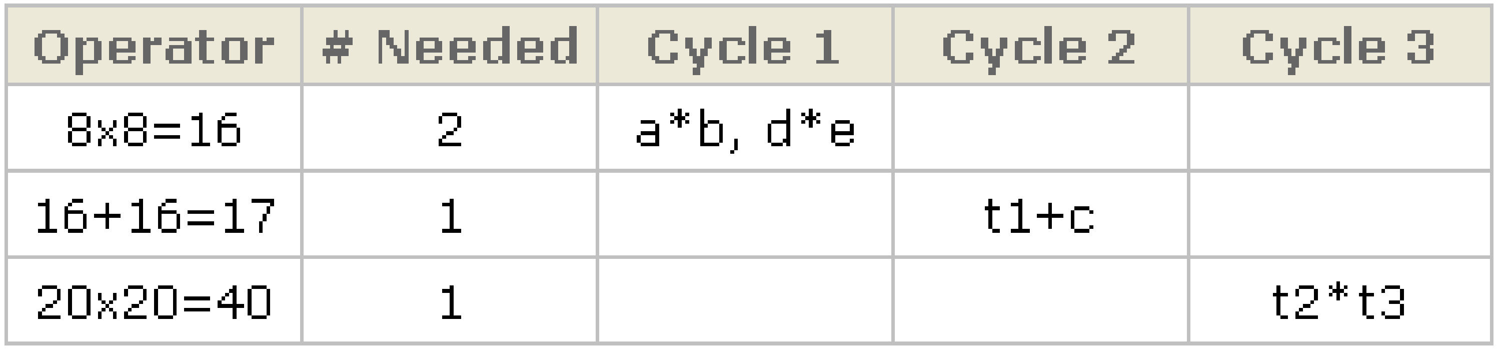
An initial allocation might be:

•Two 8x8=16 multipliers

•One 16+16=17 adder

•One 20x20=40 multiplier

- Given this allocation, if the latency of this computation were constrained to be three cycles or less, the following trivial schedule can be constructed:

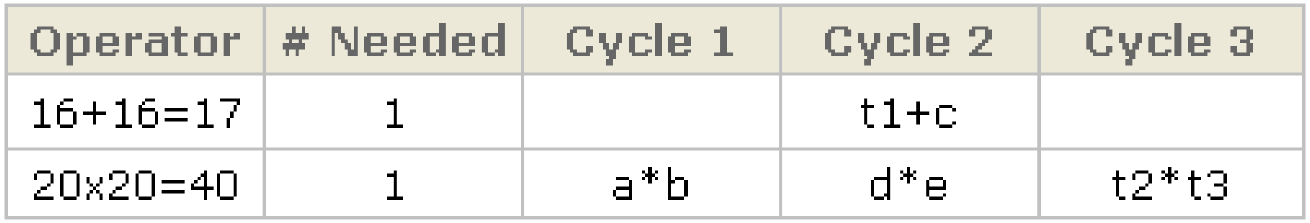


Operator	# Needed	Cycle 1	Cycle 2	Cycle 3
8x8=16	2	a*b, d*e		
16+16=17	1		t1+c	
20x20=40	1			t2*t3

- The scheduling algorithm should note that the d*e operation has “mobility” to be scheduled in either cycle 1 or cycle 2. It should then schedule it in cycle 2 in order to eliminate one 8x8 multiply operator.

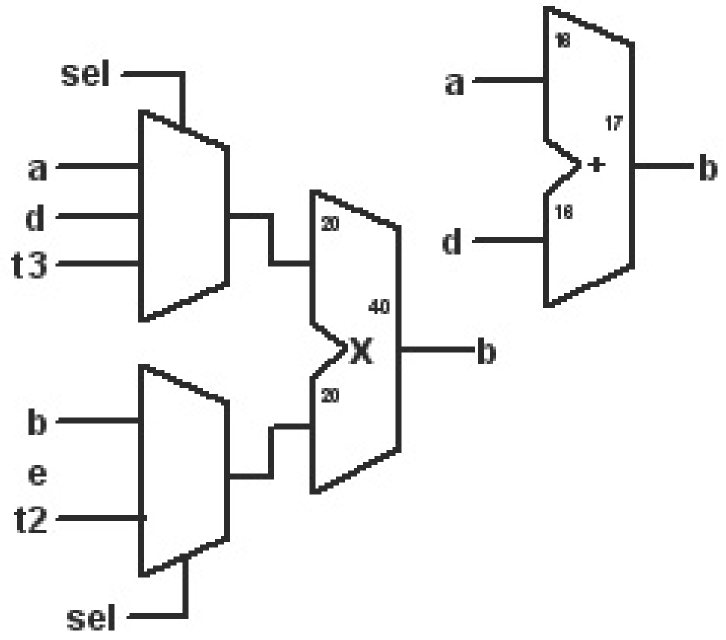
Operator	# Needed	Cycle 1	Cycle 2	Cycle 3
8x8=16	1	a*b	d*e	
16+16=17	1		t1+c	
20x20=40	1			t2*t3

The binding process now assigns these operators to specific instances of functional units. It may optimize this by noting that the 20x20 multiplier can perform the function of the 8x8 multiplier. It may also select a smaller but slower adder if the timing permits.



Operator	# Needed	Cycle 1	Cycle 2	Cycle 3
16+16=17	1		t1+c	
20x20=40	1	a*b	d*e	t2*t3

- This would imply an architecture like this:


sel
a
d
t3
b
e
t2
sel
20
20
40
X
b
a
d
18
18
17
+
b