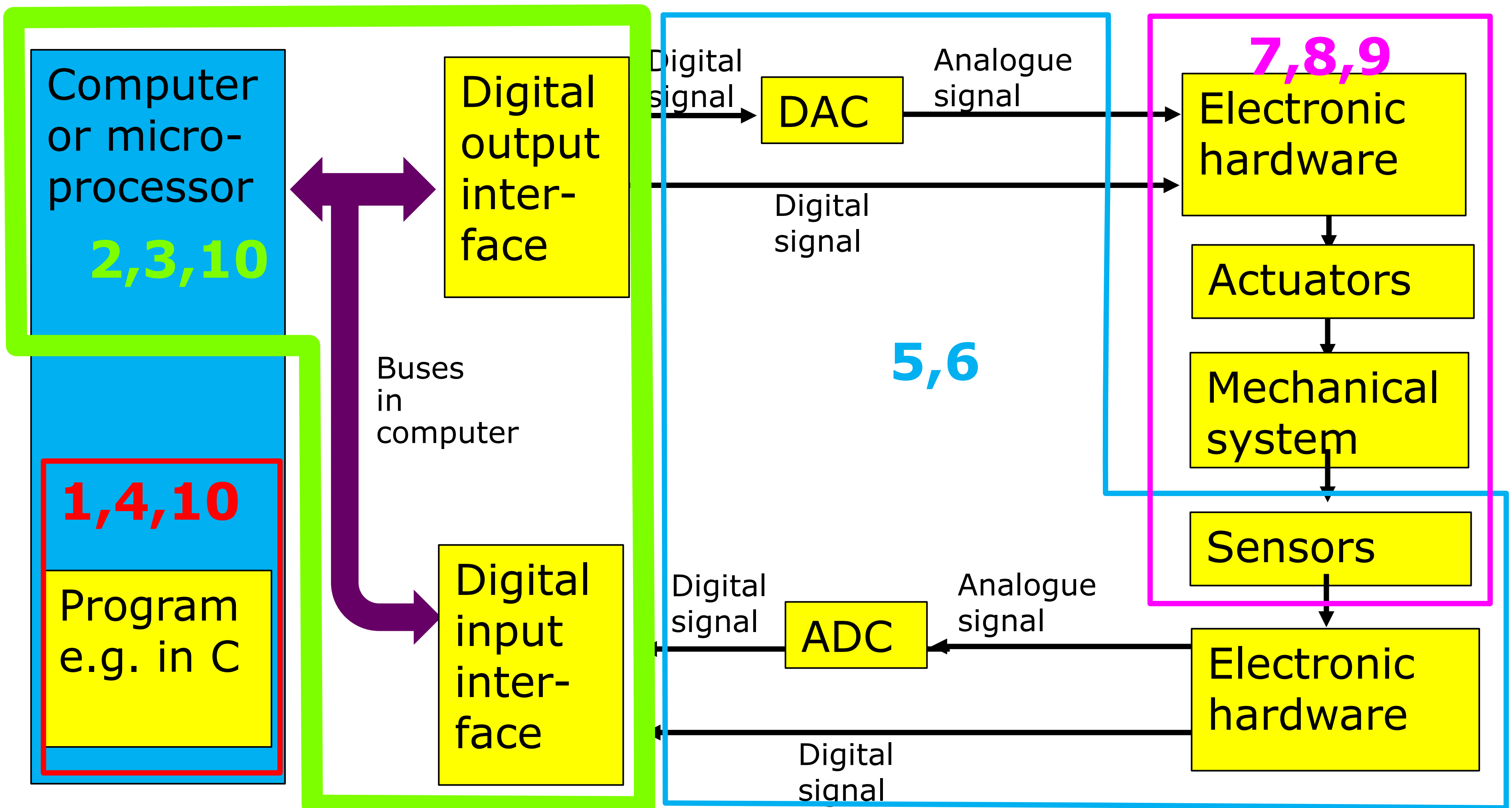
Computer Engineering and Mechatronics MMME3085

Lecture 10: Multitasking, Interrupts, FPGAs as an alternative to computing

Abdelkhalick Mohammad, AMB, B37

A typical mechatronic system

- This is where Lecture 10 fits in



The University of Nottingham

Overview of lecture

- Introduction to real-time issues
- Multitasking
- Interrupts:
  - The Arduino way
  - The low-level, AVR register way
  - Revisiting examples you’ve already seen
- FPGAs – what are they
- Lookup tables
- Applications of FPGAs

Multitasking

Introduction

Multitasking issues

- We tend to take it for granted that computers can do more than one thing at once:
  - We can run multiple applications under Windows
  - "Mainframe computers" (the 1960s-80s equivalent of servers, along with the main Unix machines at the University etc.) have large numbers of simultaneous users
  - A single microprocessor may be performing several different control functions at once

But, the question is does the computer handle these tasks in real time simultaneously?!

The University of Nottingham

Multitasking issues

- In fact a single microprocessor can only perform one job at a time

- Multitasking involves each task sharing the computing resources (microprocessors or cores) available and taking turns according to some scheduling algorithm

The University of Nottingham

Multitasking issues

- Even if many tasks appear to be happening at once (Windows clock, playing music, capturing keystrokes in Word etc.), on a single-core machine, only one task will be happening at once
- Even on a multicore machine, only (bits of) say two or four tasks will take place simultaneously

The University of Nottingham

Multitasking on the Arduino

- Most Arduinos only have a single core, so we are limited to getting our one core to multi-task
- It can't really do that, of course – we have to have some way of scheduling different aspects of our tasks are executed at the right time without delaying each other
- We will consider:
  - Simple scheduling approaches
  - Interrupts (to give urgent tasks priority)

University of Nottingham
UK | CHINA | MALAYSIA

Multitasking

Simple scheduling approaches

Cooperative multi-tasking

- One timed loop for each task
- You already know how to do this! Basis of:
  - One of the first tasks you did (blinking multiple LEDs independently)
  - Print loop and control loop running at different intervals (1000 ms and 20 ms)
- Non-pre-emptive: there is no attempt to stop each iteration of each task running, it is executed and lets another task take over
- "Cooperative" as tasks must cooperate – each task must make way for the others!

Cooperative multi-tasking – a simple implementation

```
/*
  Simple multi-tasking: blinking 2 LEDs

  Written by Arthur Jones, adapted from
  BlinkWithoutDelay example
*/

const int ledPin1= 12, ledPin2 = 13;
const long interval1 = 1000, interval2 = 100;

bool ledState1 = false, ledState2 = false;
unsigned long prevMillis1 = 0, prevMillis2 = 1;

void setup()
{
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}
```

```
void loop()
{
  unsigned long currentMillis = millis();

  if (currentMillis - prevMillis1 >= interval1)
  {
    // save the last time you blinked LED 1
    prevMillis1 = currentMillis;
    ledState1=!ledState1;
    digitalWrite(ledPin1, ledState1);
  }

  if (currentMillis - prevMillis2 >= interval2)
  {
    // save the last time you blinked LED 2
    prevMillis2 = currentMillis;
    ledState2=!ledState2;
    digitalWrite(ledPin2, ledState2);
  }
}
```

Pre-emptive multi-tasking

- Each task is allowed a certain amount of machine time to run
- Then it is put on hold, and the next task runs for a time
- Then the next task…until we get back to the first task
- It is pre-emptive because a task gets stopped to allow next one to continue
- Need way of putting task on hold: multiple stacks to store the state of each task

Pre-emptive multi-tasking e.g. round-robin scheduling

- Can be done on AVR microprocessors but much harder to do
- Won't look at this in any detail
- See:

https://www.hackster.io/AkashKollipara/preemptive-multitasking-scheduler-for-avr-e985fd

But what if we want to change our priorities to deal with something happening externally?

University of Nottingham
UK | CHINA | MALAYSIA

Responding to external events

Responding to external events

- There are two basic ways of making your computer respond to external things happening:
- **Polling:**
  - Repeatedly checking a digital input pin to see whether it has changed state etc.
  - This takes up computing time and may miss an event taking place.
  - Excellent example is encoder state machine program: missed pulses when printing to serial port

Responding to external events

- **Interrupt:** a feature of a computer which enables an internal or external signal or event to:
  - interrupt execution of the a program
  - cause the execution of special code not directly called by the main program
- Typical causes of interrupts:
  - Hardware: timer event e.g. overflow,
  - External event e.g. pin change, key stroke, mouse movement
  - Software: arithmetic overflow, zero divide

University of Nottingham
UK | CHINA | MALAYSIA

Interrupt

Introduction

What is an interrupt?

- Formal definition (from A Dictionary of Computer Science, Oxford University Press, 7th Ed, 2016)

"A signal to a processor indicating that an asynchronous event has occurred. The current sequence of events is temporarily suspended (interrupted) and a sequence appropriate to the interruption is started in its place"

Interrupt terminologies

- **Interrupt handler** or **interrupt service routine**

An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.

Interrupt terminologies

- **Interrupt vector:** misleading name, it an entry in a list or table of addresses of interrupt service routines

Table 14-1. Reset and Interrupt Vectors (Continued)

Vector No.	Program Address[2]	Source	Interrupt Definition
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052[3]	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C[3]	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C

(extract from Atmega 2560 data sheet)

Interrupts on the Arduino

Interrupts on the Arduino

- Not surprisingly, we'll consider this from two viewpoints:
  - Within the **Arduino language** (limited functionality available)
  - Via **lower-level programming** of the AVR Atmega microcontroller, specifically via **registers**:
    - Far more versatile (more flexibility)
    - Much greater functionality
    - But a bit harder!

Interrupts on the Arduino

1. Arduino language

Arduino interrupt functionality

- Arduino language itself only supports one kind of interrupt, the "**external interrupt**":
  - Limited range of pins (on Mega it's pins 2-3 and 18-21, on Uno only pins 2 & 3)
  - Separate interrupt vector for each pin
  - Interrupt is triggered when interrupt pin:
    - Is low (option given name **`LOW`**)
    - Changes state (**`CHANGE`**)
    - Goes from low to high (**`RISING`**)
    - Goes from high to low (**`FALLING`**)

Arduino interrupt functionality

- Link a function to an external interrupt event (so effectively turning it into an ISR):

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

where:

- **`ISR`** is name of function to call (must take no parameters and return nothing)
- **`mode`** is **`LOW`**, **`CHANGE`**, **`RISING`** or **`FALLING`**

Example

- The example we used was to call the function **`updateEncoderStateMachine`** when pins 2 or 3 (named **`channelA`** and **`channelB`**) changed state:

```
attachInterrupt(digitalPinToInterrupt(channelA), updateEncoderStateMachine, CHANGE);
attachInterrupt(digitalPinToInterrupt(channelB), updateEncoderStateMachine, CHANGE);
```

- This effectively turned **`updateEncoderStateMachine`** into an interrupt service routine without doing any register-level programming

Arduino interrupt functionality

Pin change interrupts:

- Other pins can be used to generate interrupts using (for example) the **pinChangeInterrupt** library

- Broadly similar functionality to native Arduino functions for external interrupts
- but only for **RISING**, **FALLING** or **CHANGE**:

```
attachPCINT(digitalPinToPCINT(pin),ISR, mode);
```

Arduino interrupt functionality

Timer interrupts

- Powerful functionality can be obtained using interrupts triggered by timer events such as:
  - Counter overflow (when counter exceeds maximum value and rolls over to 0)
  - Counter compare match (when counter value matches a predetermined value)
- Libraries available: **TimerOne**, **TimerThree**
- Can generate frequencies, non-standard PWM signals, call ISRs at specified intervals etc.

University of Nottingham
UK | CHINA | MALAYSIA

Interrupts on the Arduino

2. The AVR way

Interrupts – the AVR way

- It will be no surprise that we can get the full power of interrupts on AVR chips by:
  - Reading the datasheet carefully!
  - Configuring individual **control registers** to set up interrupts
  - Making use of the **interrupt vector** for each kind of interrupt in order to call the interrupt service routine (**ISR**)
- Best illustrated via examples from your experience – seen previously in labs, but probably not understood!

University of Nottingham
UK | CHINA | MALAYSIA

Interrupts on the Arduino

2. The AVR way: Example 1 - Encoder state machine

Examples of AVR interrupts: Encoder program – the AVR way

- Alternative implementation of the interrupt-driven encoder state machine function: ISR to update state m/c when pin 2 or 3 changes
- “External interrupts”, e.g. pins 2&3 on Mega:
  - Pin 2 linked to interrupt vector **`INT4_vect`**
  - Pin 3 linked to interrupt vector **`INT5_vect`**

  (on Uno it’s **`INT0_vect`** and **`INT1_vect`**)

Examples of AVR interrupts: Encoder program – the AVR way

- Alternative implementation of the interrupt-driven encoder state machine function: ISR to update state m/c when pin 2 or 3 changes
- “External interrupts”, e.g. pins 2&3 on Mega:
  - Pin 2 linked to interrupt vector **`INT4_vect`**
  - Pin 3 linked to interrupt vector **`INT5_vect`**

    (on Uno it’s **`INT0_vect`** and **`INT1_vect`**)

1. Need to configure interrupts using registers
2. Then we just define our routine e.g. as

```
void ISR(INT4_vect)
```

1. Configure interrupts via registers

- Configure the interrupts using External Interrupt Control Registers EICRA and EICRB (INT7:4)

EICRB – External Interrupt Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-3. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request
1	1	The rising edge between two samples of INTn generates an interrupt request

Note: 1. n = 7, 6, 5 or 4.

2. Define our routine

- Then we can (re)enable external interrupts by setting bits 4 and 5 in register EIMSK

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Then we just define our interrupt service routine to update state machine e.g. as

```
void ISR(INT4_vect)
```

- Also have code to link same ISR to **INT5_vect**

Understand underlying concept, don't learn details

Encoder program – the AVR way

- So here is our code to configure interrupts:

```
EICRB = 0;                  /* clear control register covering INT4&5 */
EICRB |= (1 << ISC40);      /* trigger INT4 when pin 2 changes */
EICRB |= (1 << ISC50);      /* trigger INT5 when pin 3 changes */
EIMSK |= (1 << INT4);       /* enable INT4 */
EIMSK |= (1 << INT5);       /* enable INT5 */
```

- And here is the start of our ISR, otherwise same as **`updateEncoderStateMachine()`**:

```
  ISR(INT4_vect)
/* User written code to update state and increment count of state machine  */
{
  channelAState = digitalRead(channelA);
  channelBState = digitalRead(channelB);
  switch (state)
  {
```

University of Nottingham
UK | CHINA | MALAYSIA

Interrupts on the Arduino

2. The AVR way: Example 2 - Counting overflows

Timer/counter interrupts the AVR way: counting overflows

- In Lab 1/consolidation session 3 we tried using Timer 5 as a counter
- Counted how many pulses we got on pin 47, tried seeing if we could use it with encoder
- No good because only counts in one direction, not an up/down quadrature counter!
- But even worse, it only counts up to 65535 then rolls over back to zero ("overflows"), so how did we overcome this?
- Answer: used an ISR to count the overflows!
- Overflow triggers vector **TIMER5_OVF_vect**

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

This ISR is called each time timer/counter 5 overflows, increments no. of overflows in **`bigLaps`**

University of Nottingham
UK | CHINA | MALAYSIA

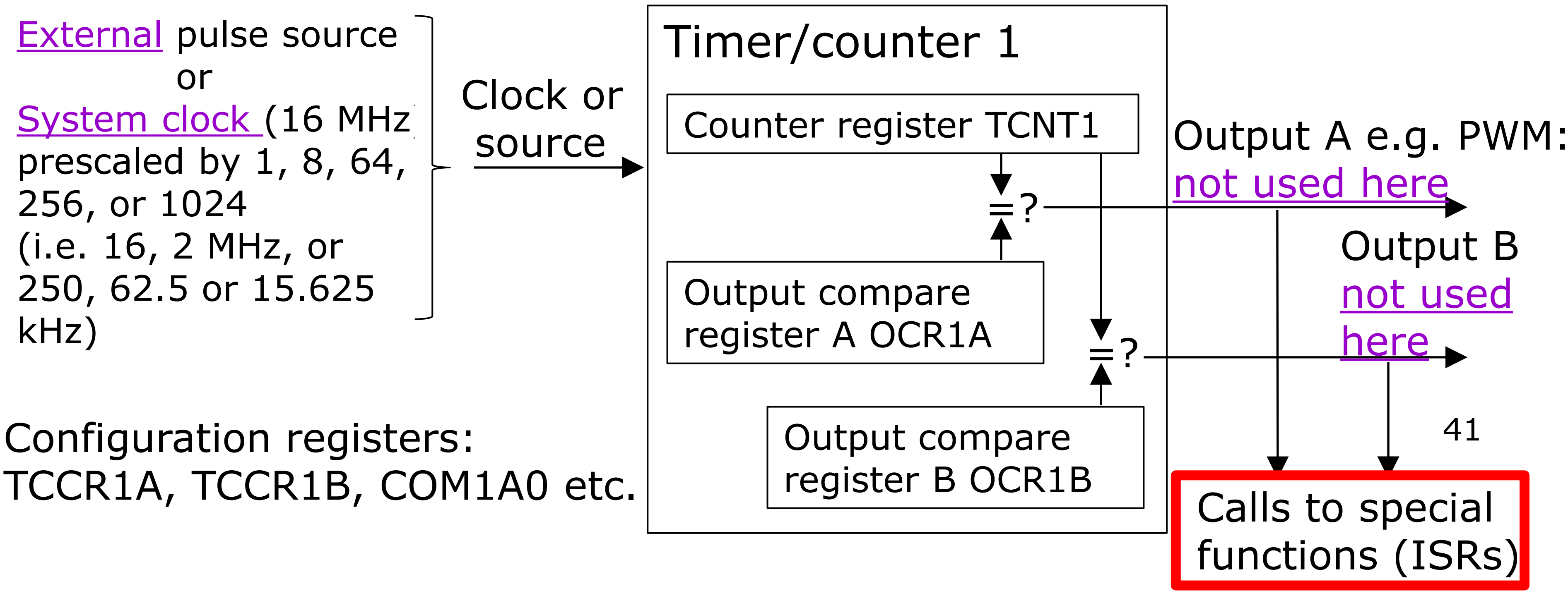
Interrupts on the Arduino

2. The AVR way: Example 3 - Stepper program

Remember this from lecture 3?

- Slightly modified from L3, simplified timer…
  (understand concept, don't learn)

Select from:

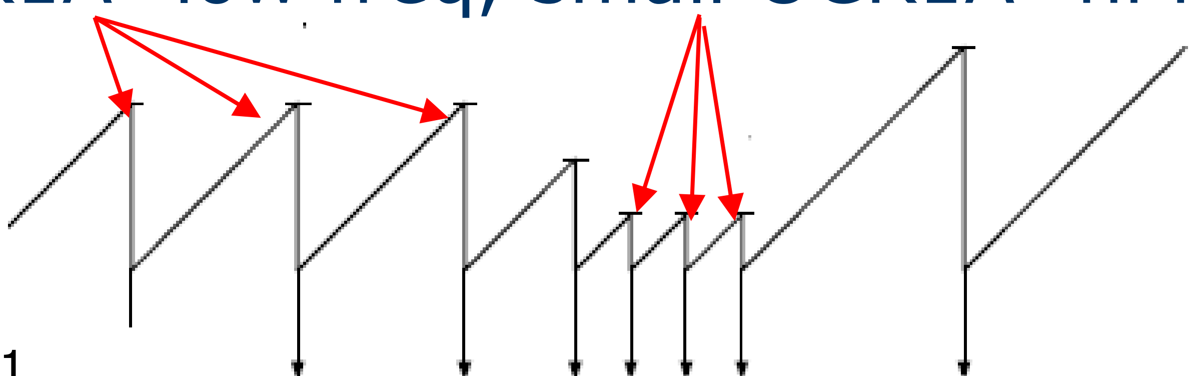


Timing diagram from Atmega 2560 datasheet

Remember this from lecture 3?

- Value (TCNT1) in the counter register increases until it reaches the value in output compare register (here OCR1A)
- Causes interrupt, calls ISR, resets TCNT1 to 0
- Big OCR1A=low freq, small OCR1A=hi freq

Value in main counter register TCNT1



ISR called & TCNT1 reset
on each "match"

ISR called at low freq

ISR called at high freq

Time per step: Implementation 2

- Alternative approach implemented in Lab 2:
  - A **hardware timer** configured in CTC mode with interval *p*, triggers ISR every *p* ticks:

```
cli();                       /* Temporarily disable interrupts */
TCCR1A = 0;                  /* No output compare */
TCCR1B = (1 << WGM12);       /* CTC mode: reset timer when TCNT1 == OCR1A */
OCR1A = 0;                   /* Set to zero initially, over-wtite in ISR */
TCCR1B |= (1 << CS12);       /* Prescaler 256 (illustrative only) */
TIMSK1 |= (1 << OCIE1A);     /* Interrupt to call ISR when TCNT1 == OCR1A */
sei();                       /* Re-enable interrupts */
```

(**don't learn details** but understand that timer triggers interrupt calling the ISR every period p)

Time per step: Implementation 2

- This **hardware timer** triggers an interrupt which is serviced by an ISR, which makes step & recalculates time *p* per step

```
ISR(TIMER1_COMPA_vect)
/* Interrupt service routine which calls moveOneStep and computeNewSpeed. */
{
  if (p == 0)
    TIMSK1 &= !(1 << OCIE1A); /* Disable interrupt if not stepping */
  else
    moveOneStep();

  OCR1A = (long)p - 1; /* Set timer (CTC) interval which is p ticks */
  computeNewSpeed(); /* Calculates timer interval p set in next ISR call */
}
```

Actually make step pulse

New interval *p* written to timer

p is re-calculated here for next step

The University of Nottingham

Important message regarding ISRs:

- When an ISR is running, the main program isn't
- So everything else stops while the interrupt is serviced
- The moral is: **keep the code in your ISR very quick and simple** so it does not greatly interfere with program flow
- Otherwise your program may "lock up", just as your interrupt-driven encoder program did when servicing too many interrupts

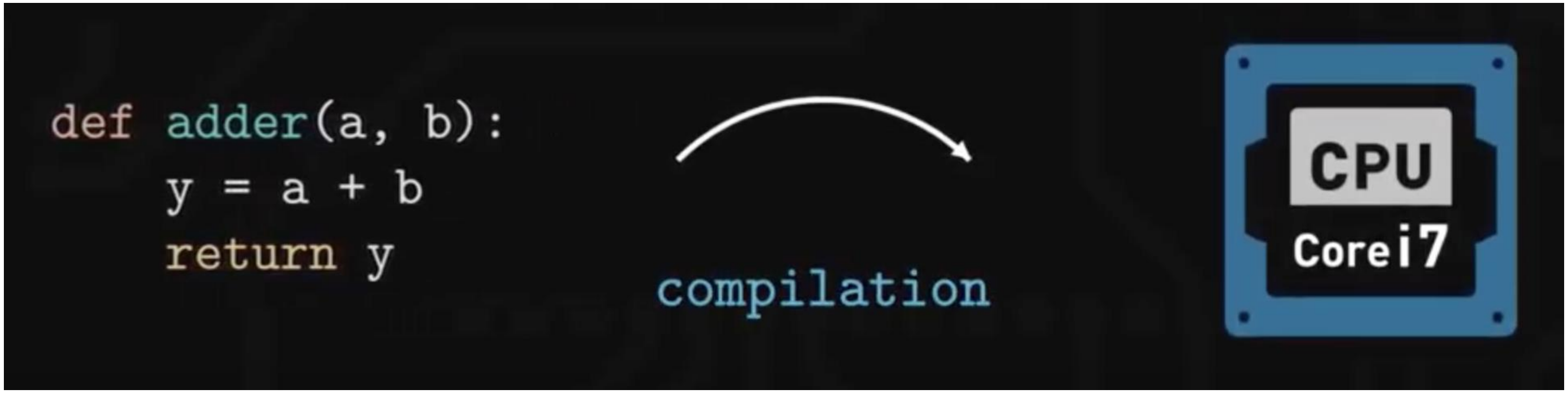
University of Nottingham
UK | CHINA | MALAYSIA

Field Programmable Gate Arrays FPGA

Introduction

What is an FPGA?

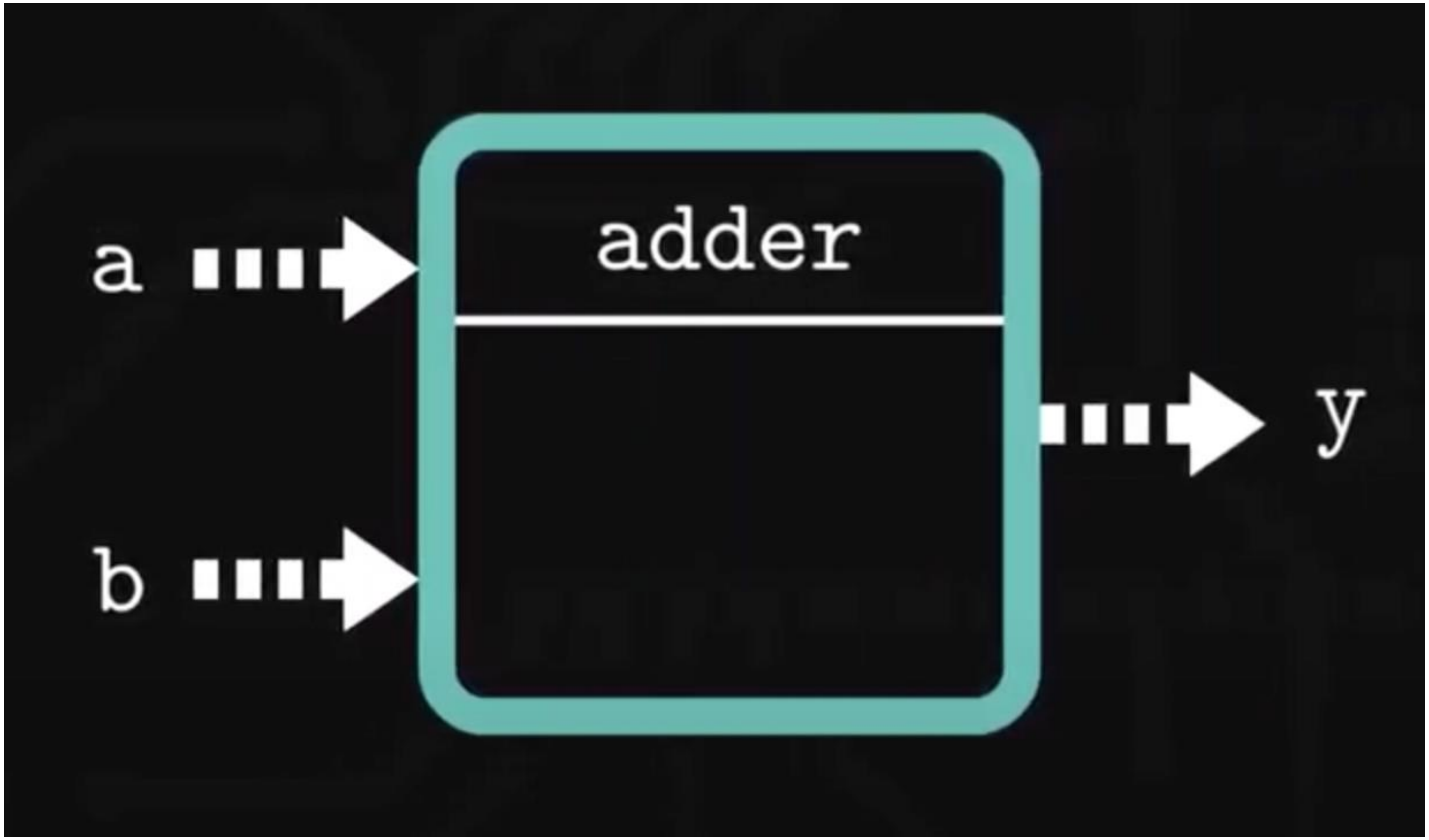
This is how the conventional microprocessor works!



Revised from a video https://www.youtube.com/watch?v=iHg0mmIg0UU

What is an FPGA?

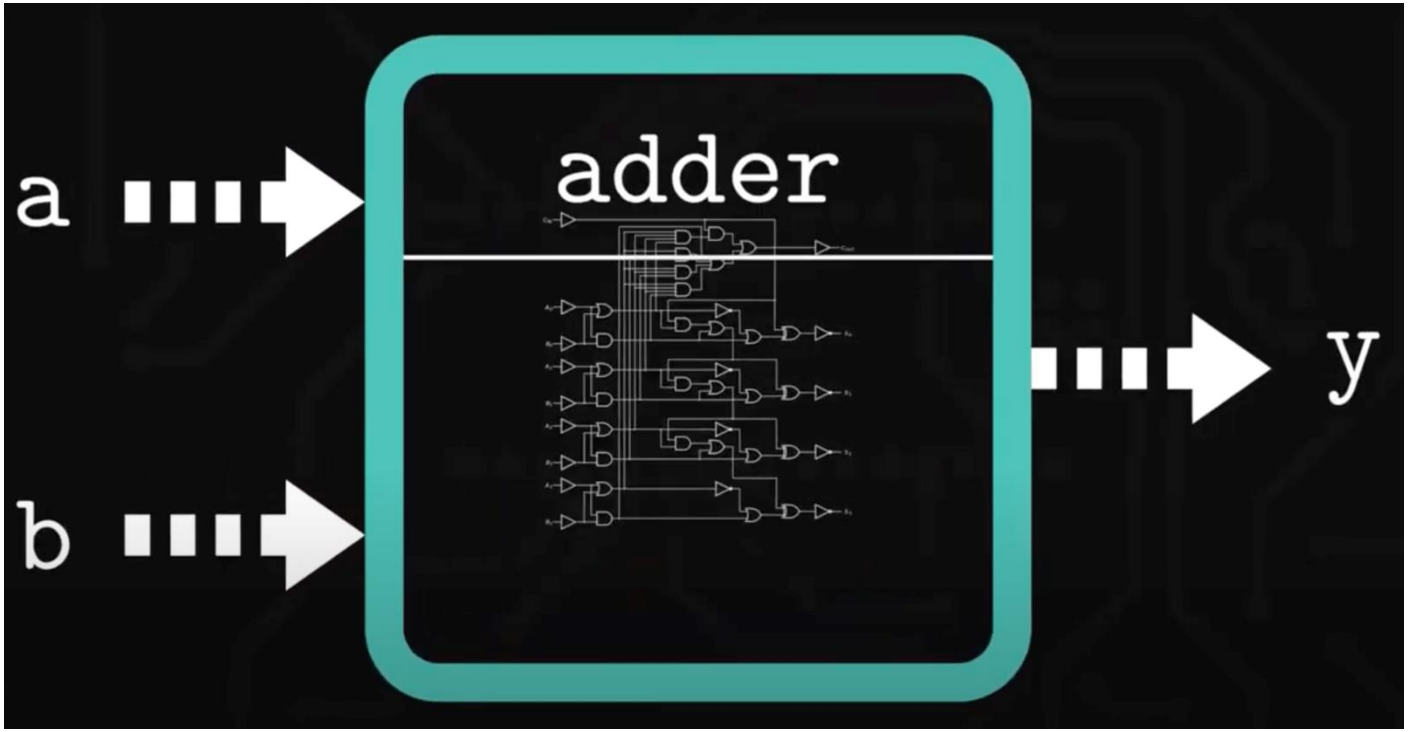
This is how the FPGA does the same job!



The added is a <u>real</u> hardware !!

What is an FPGA?

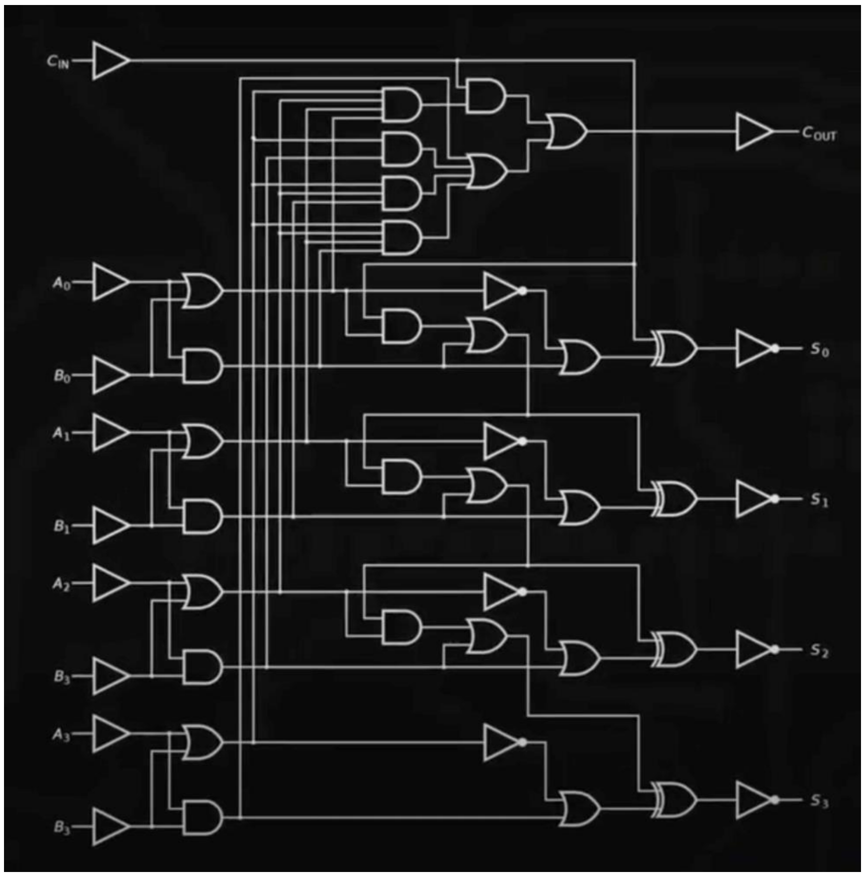
How the adder looks like inside?!



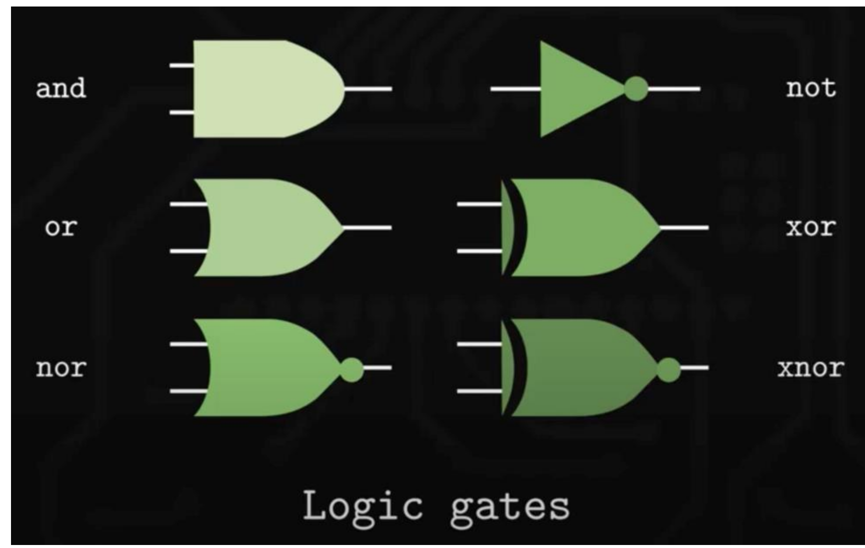
The adder consists of logic gates (as we learned previously) !!

What is an FPGA?

The University of Nottingham

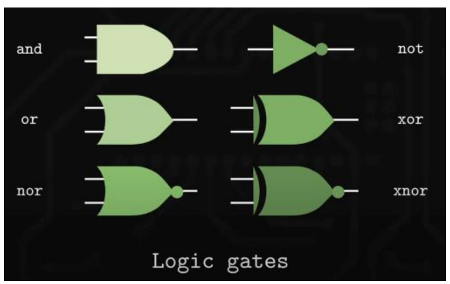


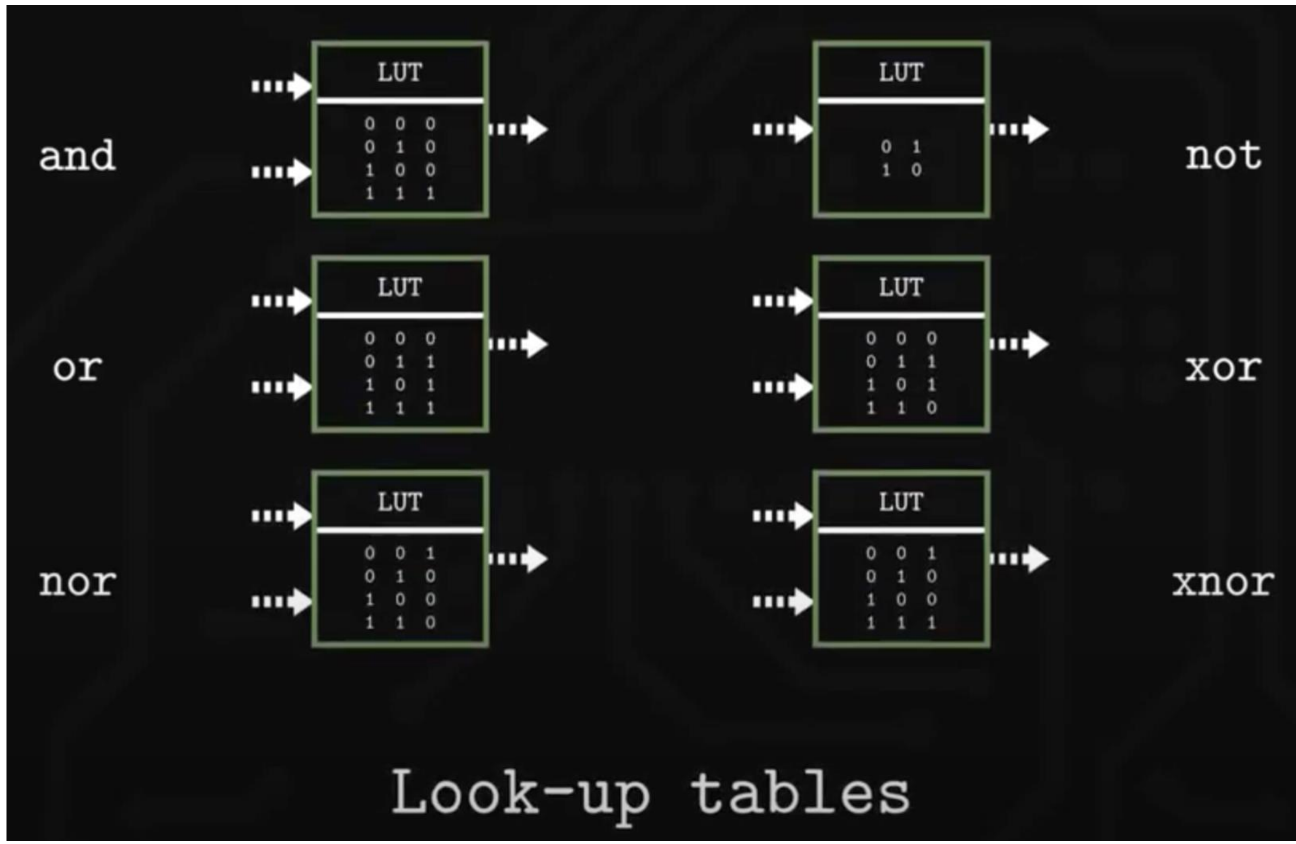
Basic logic gates



What is an FPGA?

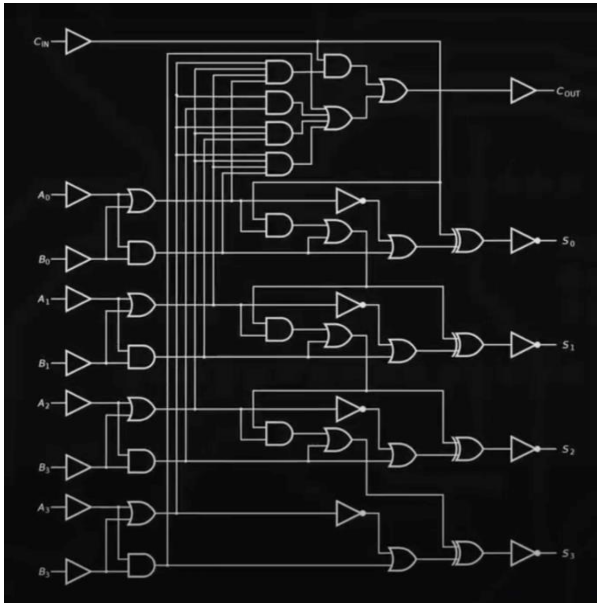
Convert
logic gates
into LUT

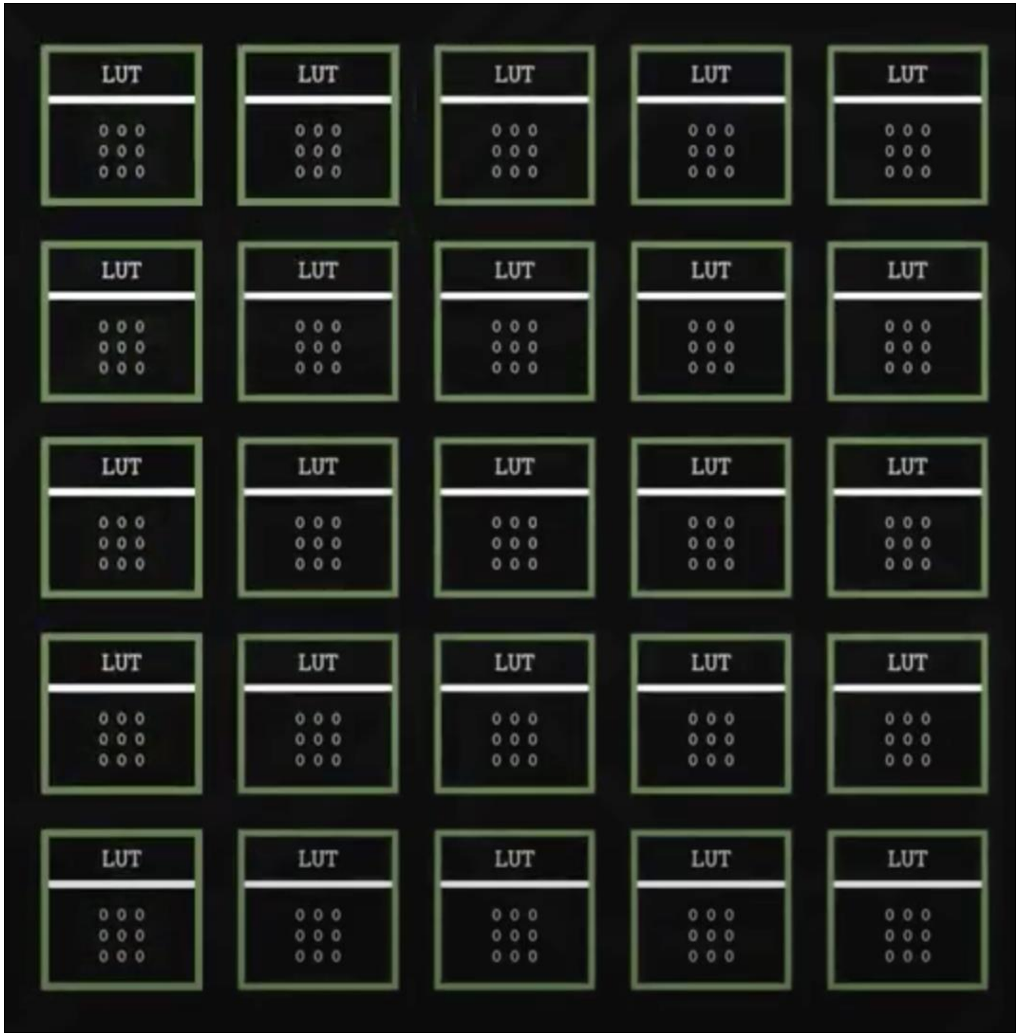




We will give an example later to show this process !

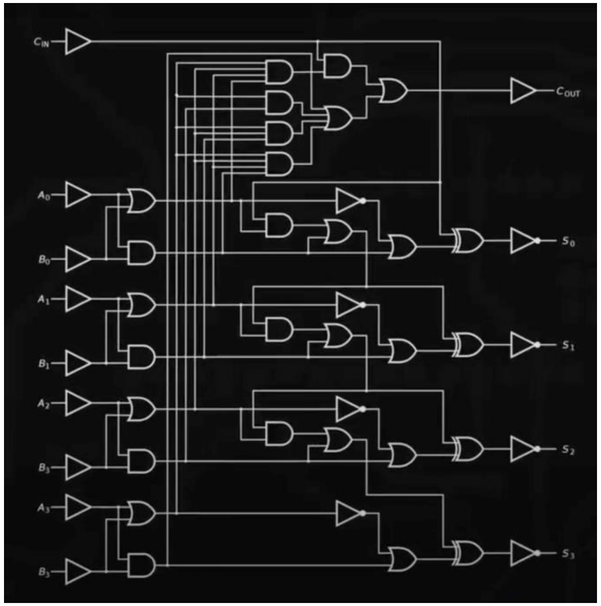
What is an FPGA?

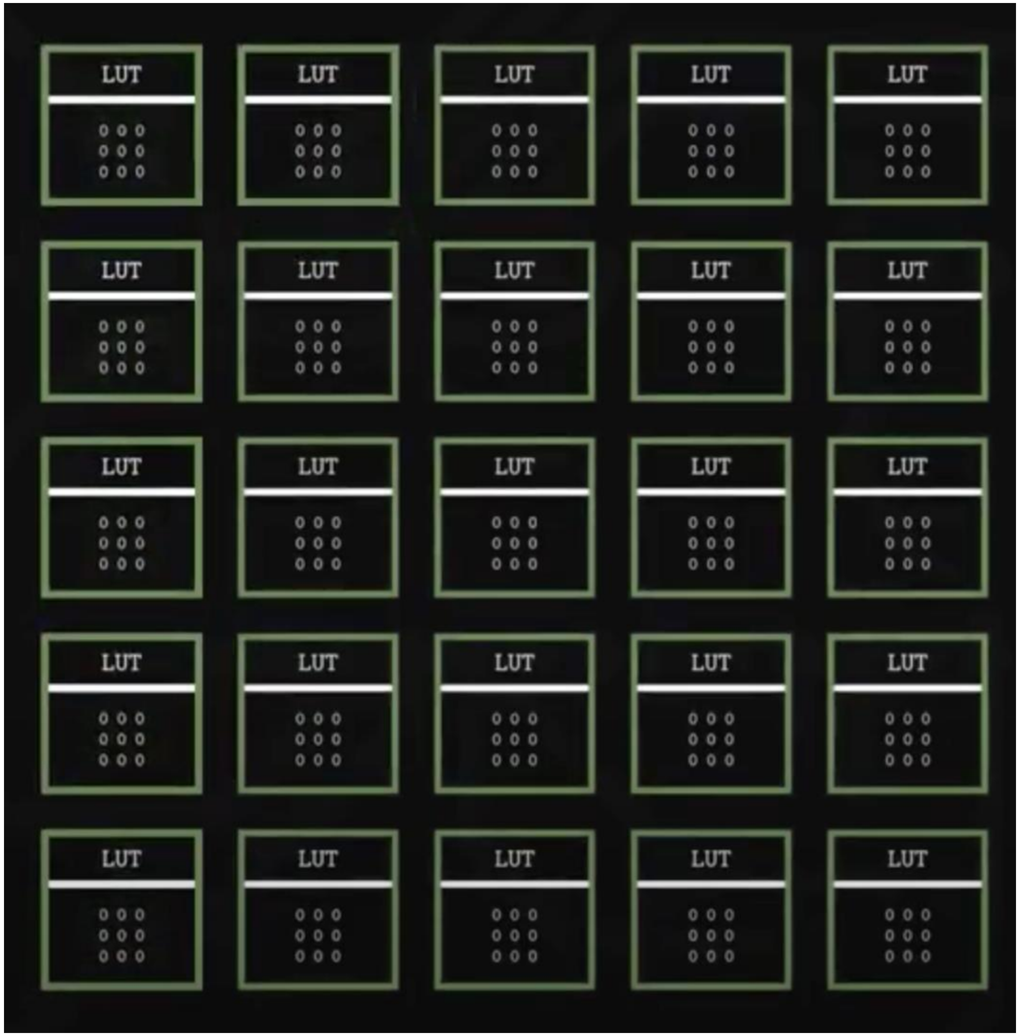




Now the adder can be converted into LUTs!

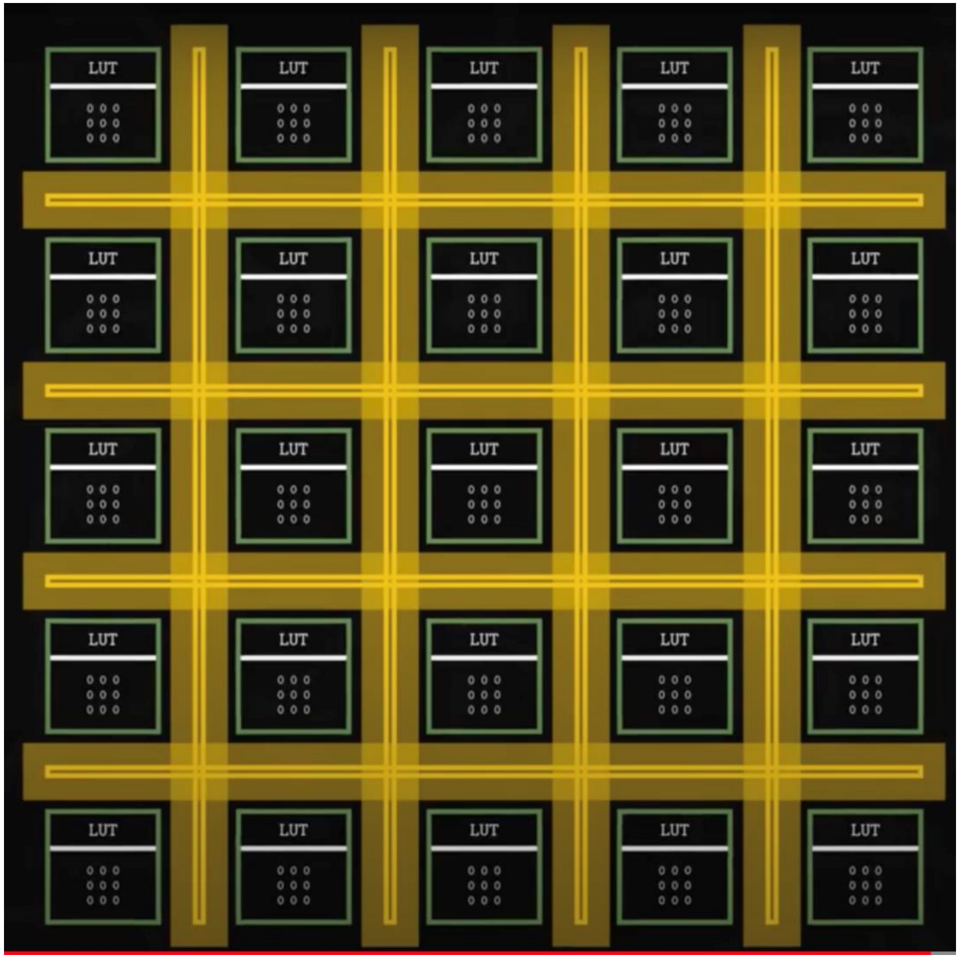
What is an FPGA?





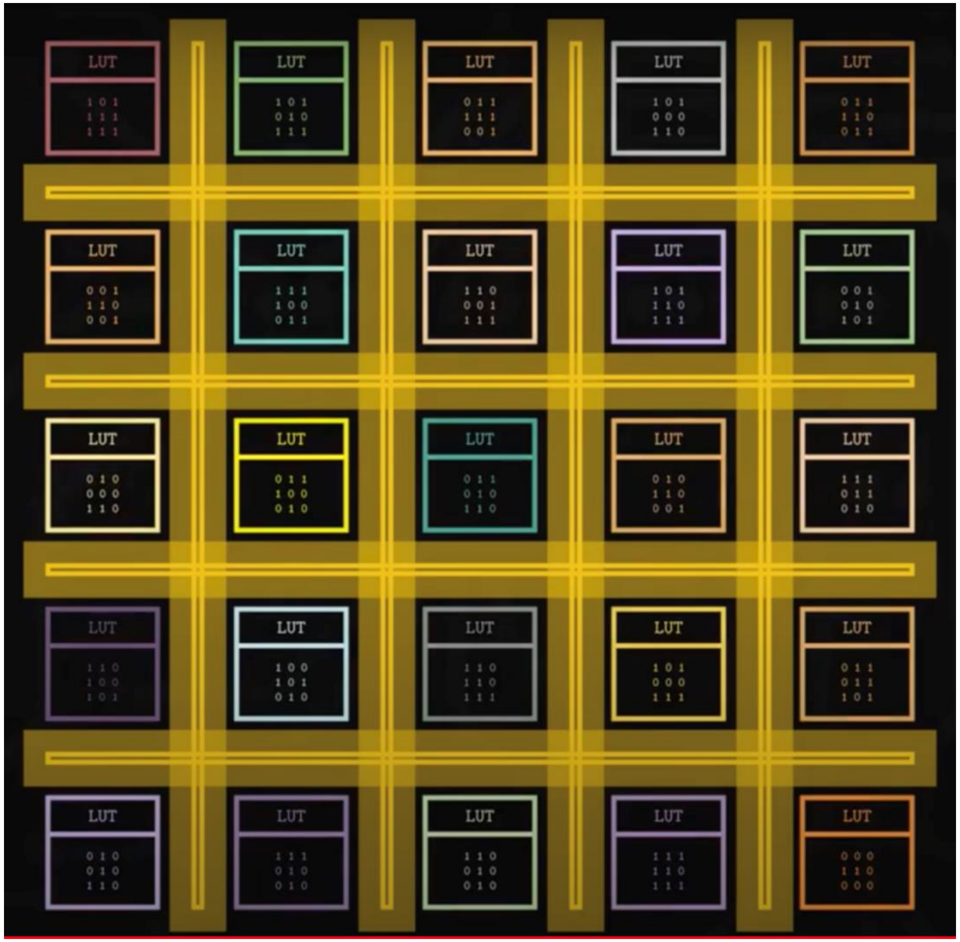
We need to connect this LUTs together !!

What is an FPGA?



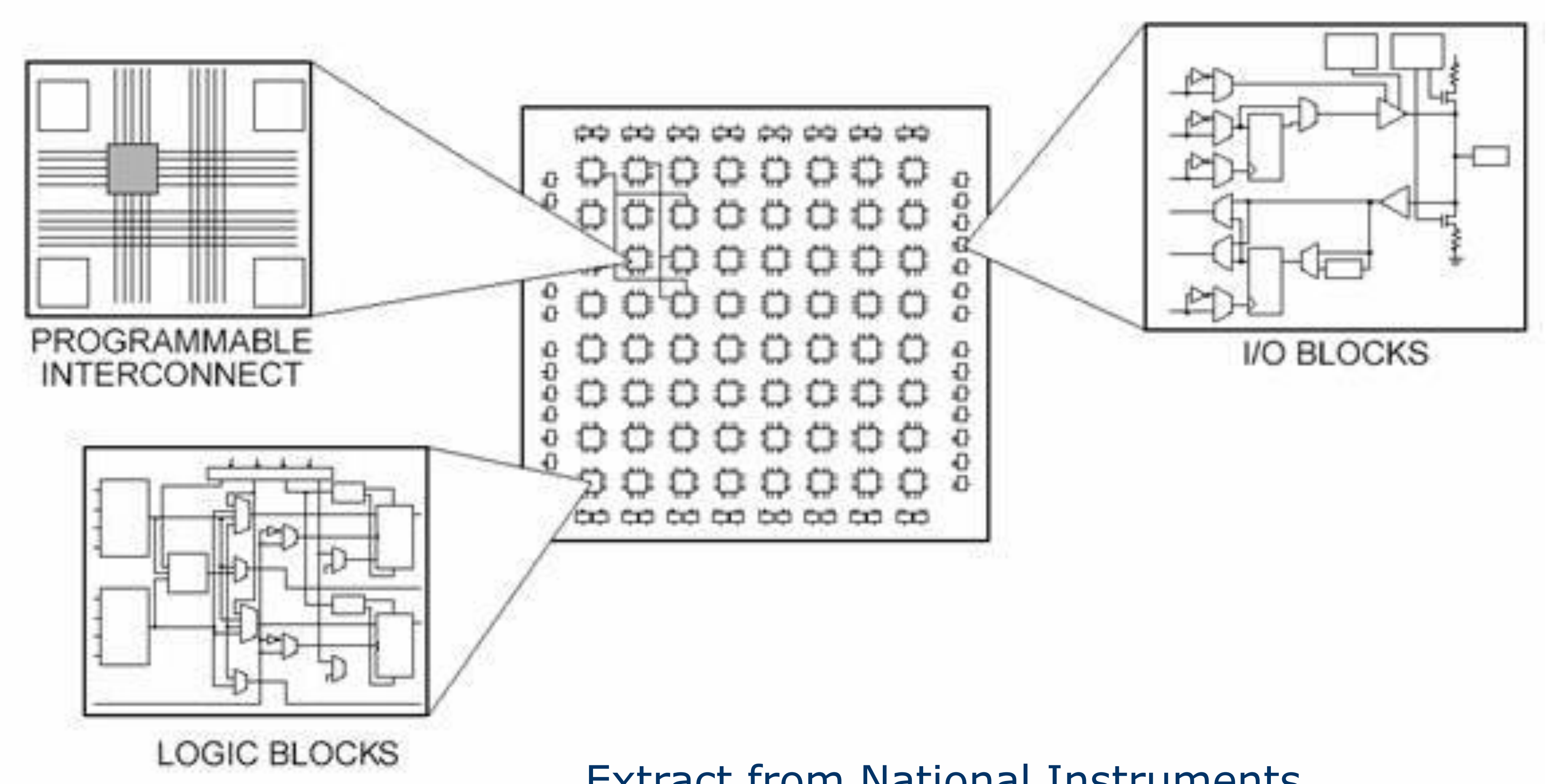
This can be done through routing fabric between the LUTs

What is an FPGA?



Now by programming the LUTs and re-wiring them, we can do any combinations of arithmetic or complex process!!

What is an FPGA?



Extract from National Instruments training materials

University of Nottingham
UK | CHINA | MALAYSIA

Field Programmable Gate Arrays FPGA

LUT

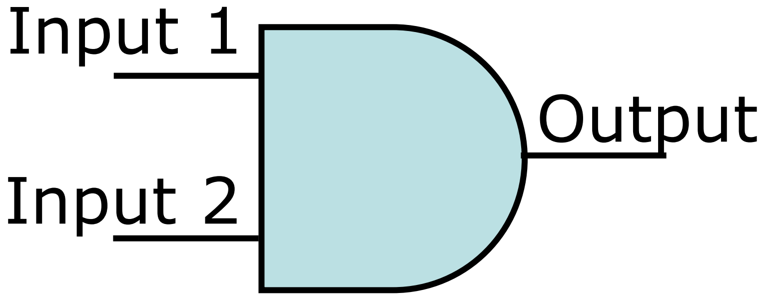
How does it work?

- In general, each logic gate is emulated using a **look-up table**

- Instead of using hard-wired logic, it codes up truth table as the **output** for each **combination** of inputs treated as a binary number

- Let's take a simple example…

Example of logic gate using FPGA



Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

- Nothing new so far…

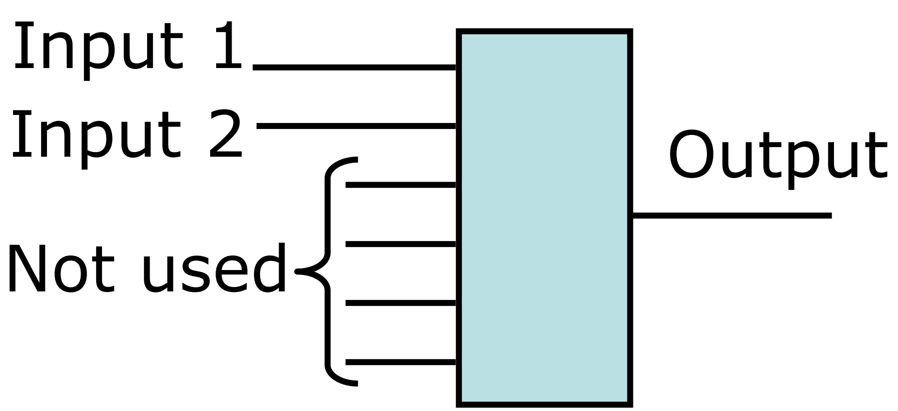
Example of logic gate using FPGA

- Treat combination of inputs as binary no.
- Store each of these binary numbers and its associated output as a table

Input 1	Input 2	Row no. in table (LUT index)		Output
		Binary	Decimal	
0	0	00	0	0
0	1	01	1	0
1	0	10	2	0
1	1	11	3	1

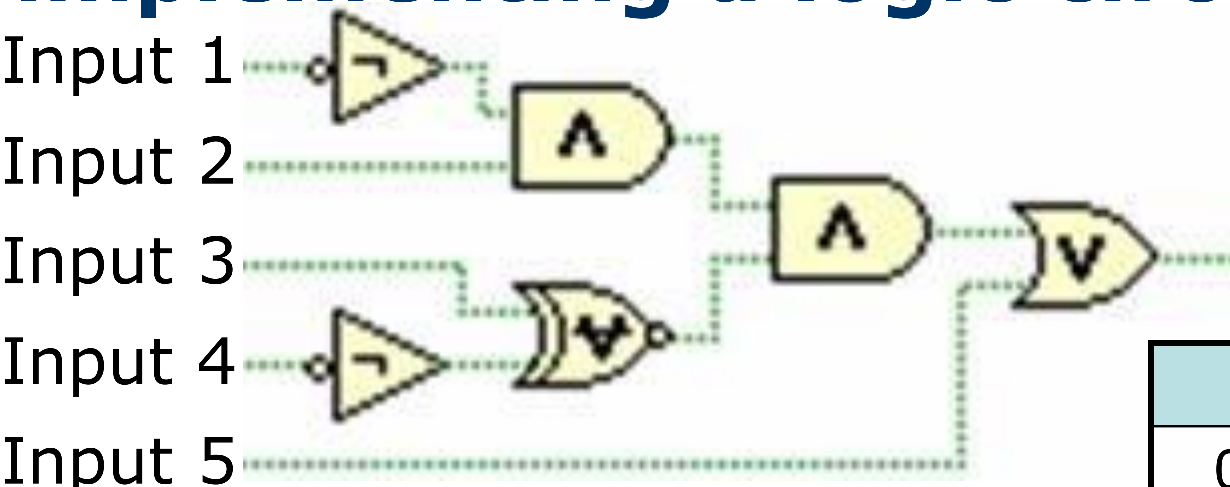
Example of logic gate using FPGA

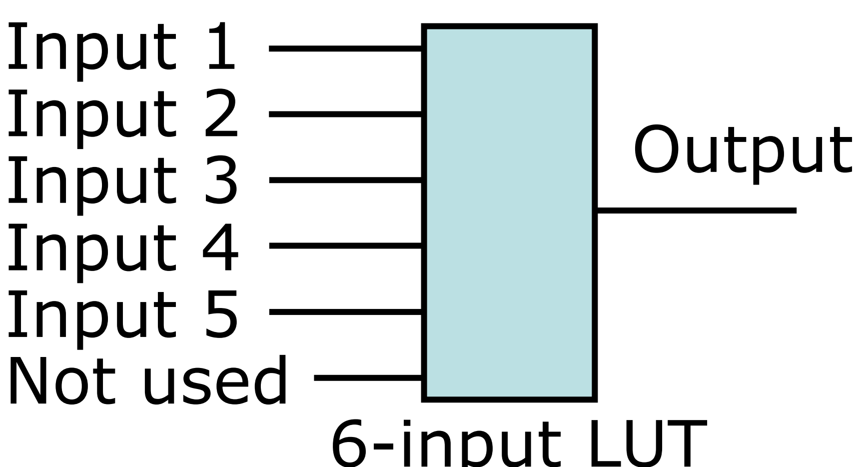
- Now have generic "lookup table" unit with:
  - several inputs (6 on Virtex 5 FPGAs), one output
  - data storage for the lookup table itself
  - logic to give output for given row of table



Index	Output
0 (00)	0
1 (01)	0
2 (10)	0
3 (11)	1
(another 60 rows unused)	

Example of a lookup table implementing a logic circuit





Index	Output
0 (00000)	0
1 (00001)	1
2 (00010)	0
3 (00011)	1
4 (00100)	0
etc. etc. - another 27 rows; 32 unused	

University of Nottingham
UK | CHINA | MALAYSIA

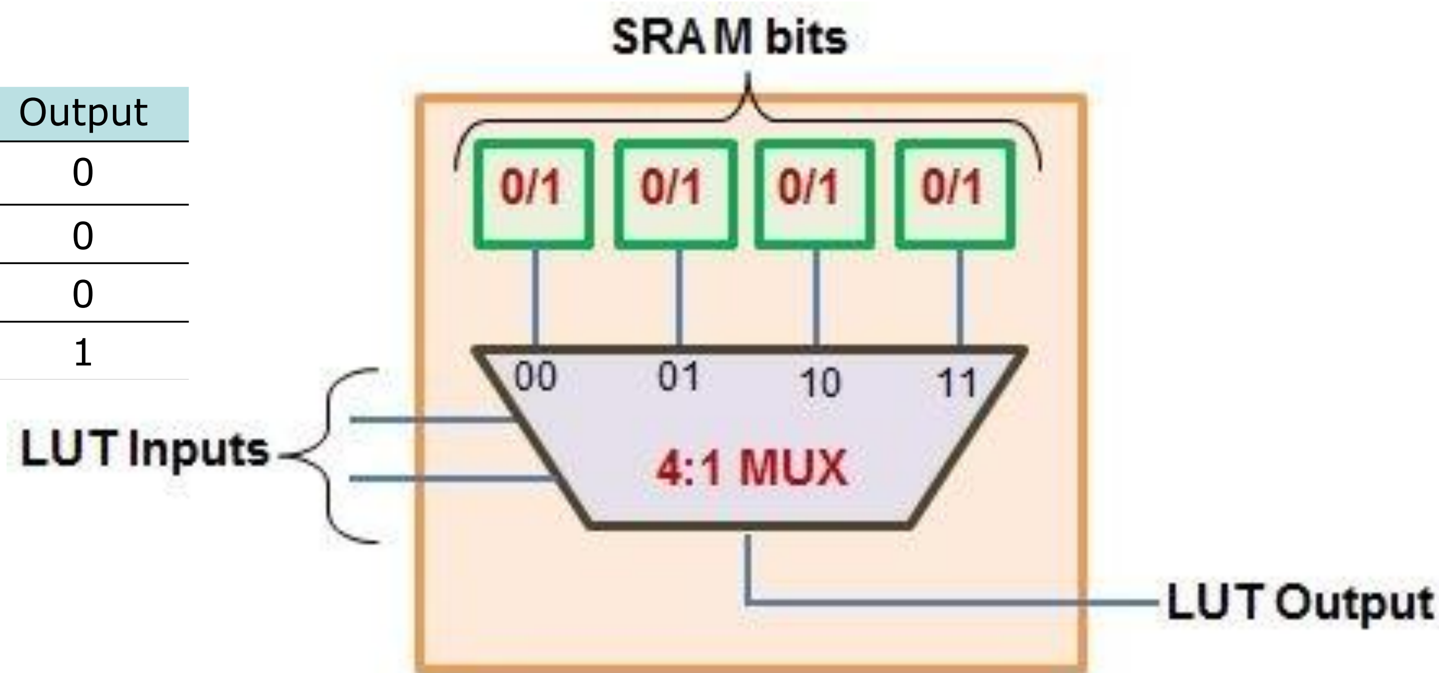
Field Programmable Gate Arrays FPGA

LUT implementation

How can we implement LUT?!

- LUTs comprise of 1-bit memory cells (programmable to hold either '0' or '1') and a set of multiplexers.
- One value among these SRAM bits will be available at the LUT's output depending on the value(s) fed to the control line(s) of the multiplexer(s).

Index	Output
0 (00)	0
1 (01)	0
2 (10)	0
3 (11)	1



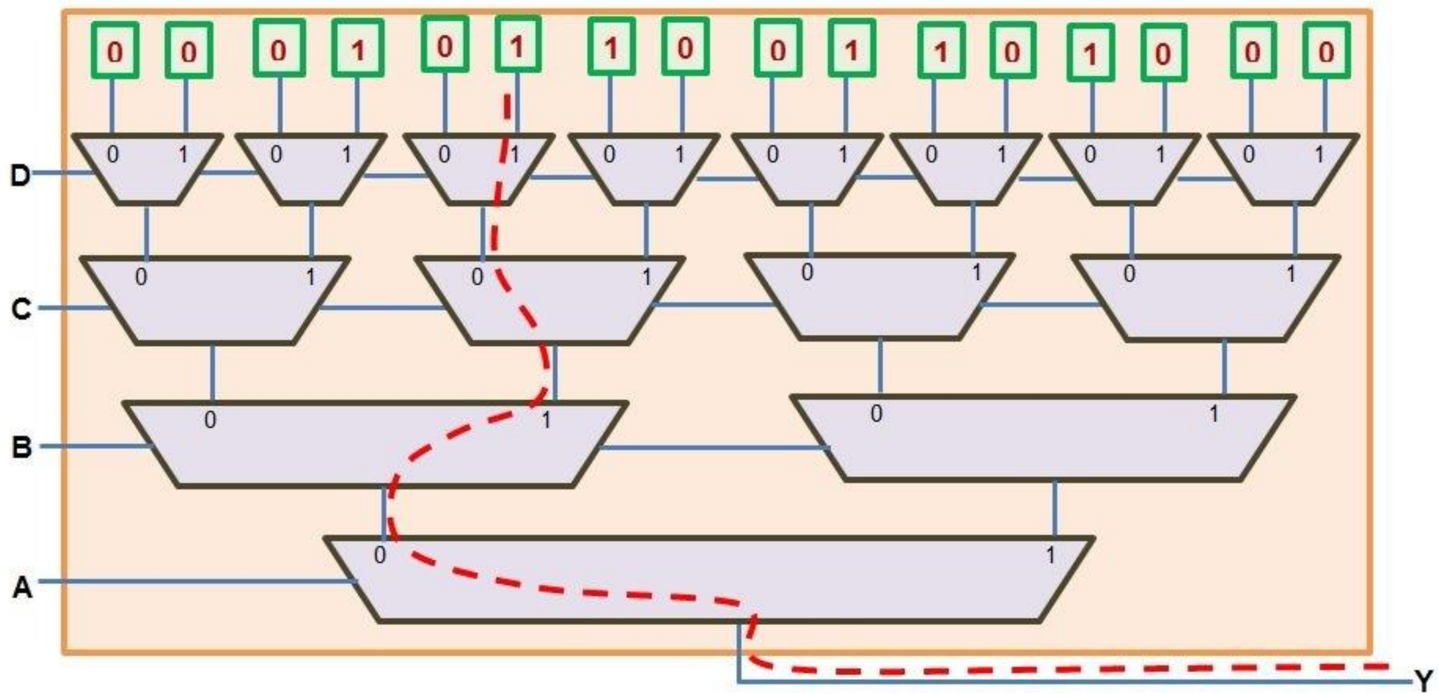
www.allaboutcircuits.com

How can we implement LUT?!

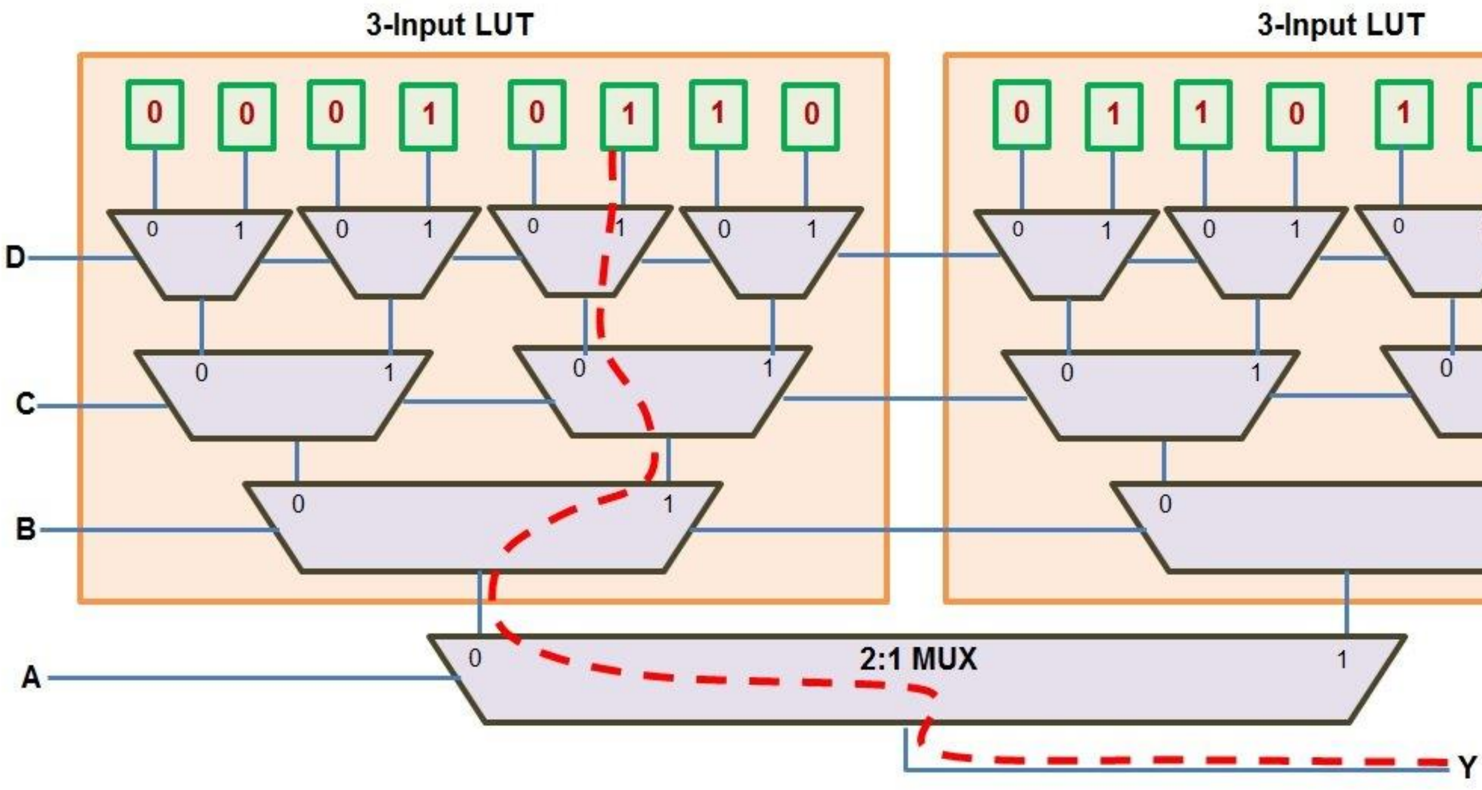
Example of 4-inputs LUT

Truth Table

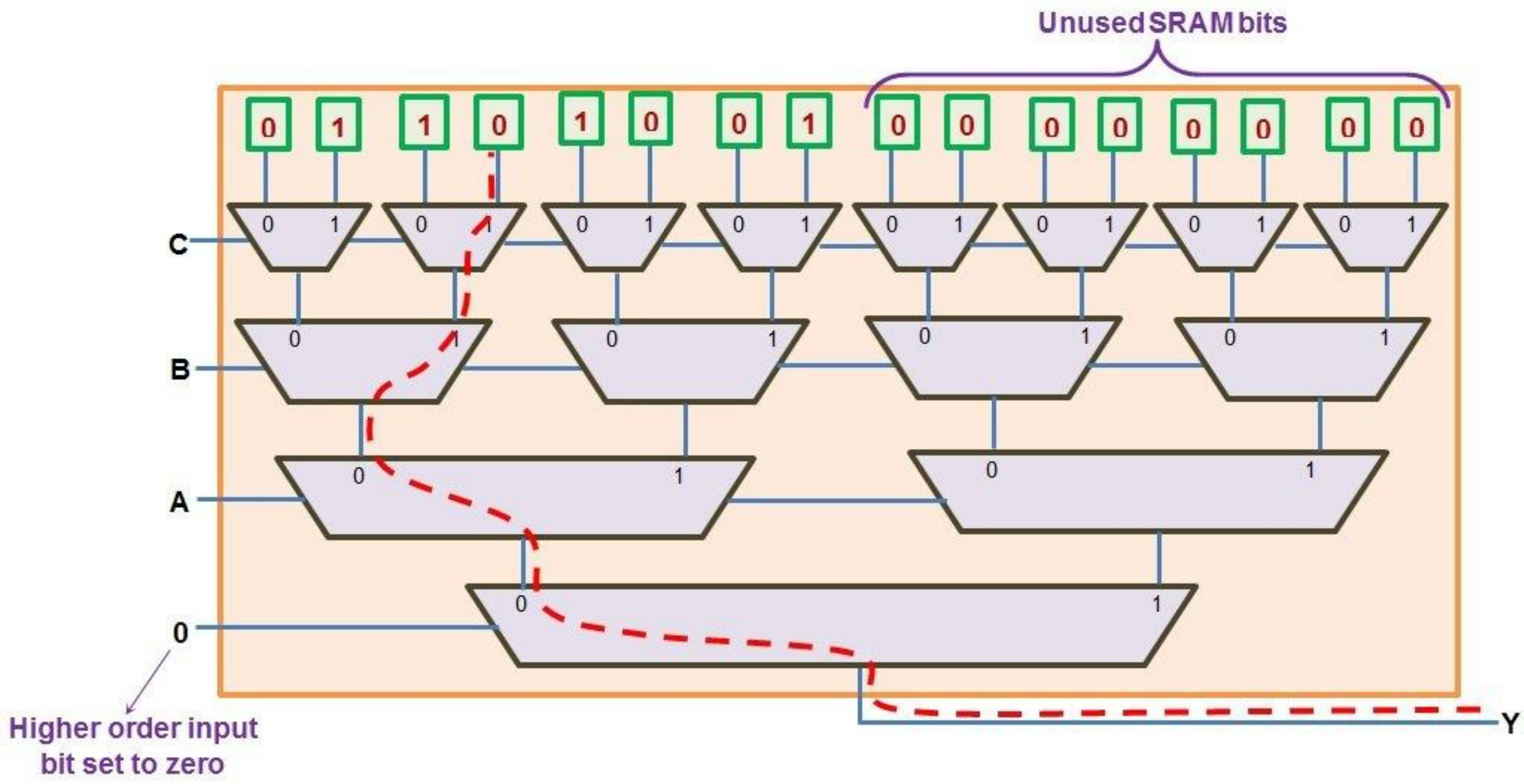
Inputs				Output
A	**B**	**C**	**D**	**Y**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Can I use 3-iputs LUTs as 4-inpts LUT?!



Can I use 4-iputs LUT as 3-inpts LUT?!



University of Nottingham
UK | CHINA | MALAYSIA

Field Programmable Gate Arrays
FPGA

FPGAs in practice

How are FPGAs programmed ?

- Conventional way to program them is via dedicated hardware development languages e.g. VHDL – not easy to learn!

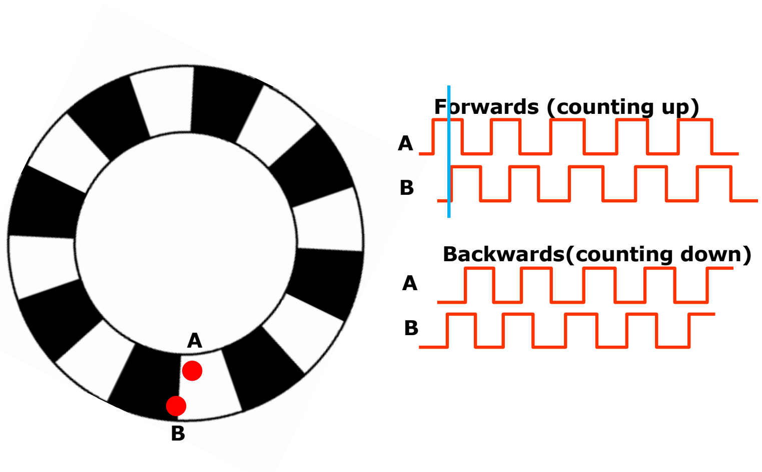
- Can alternatively use special versions of other languages e.g. C, LabVIEW, in similar manner to programming a microprocessor

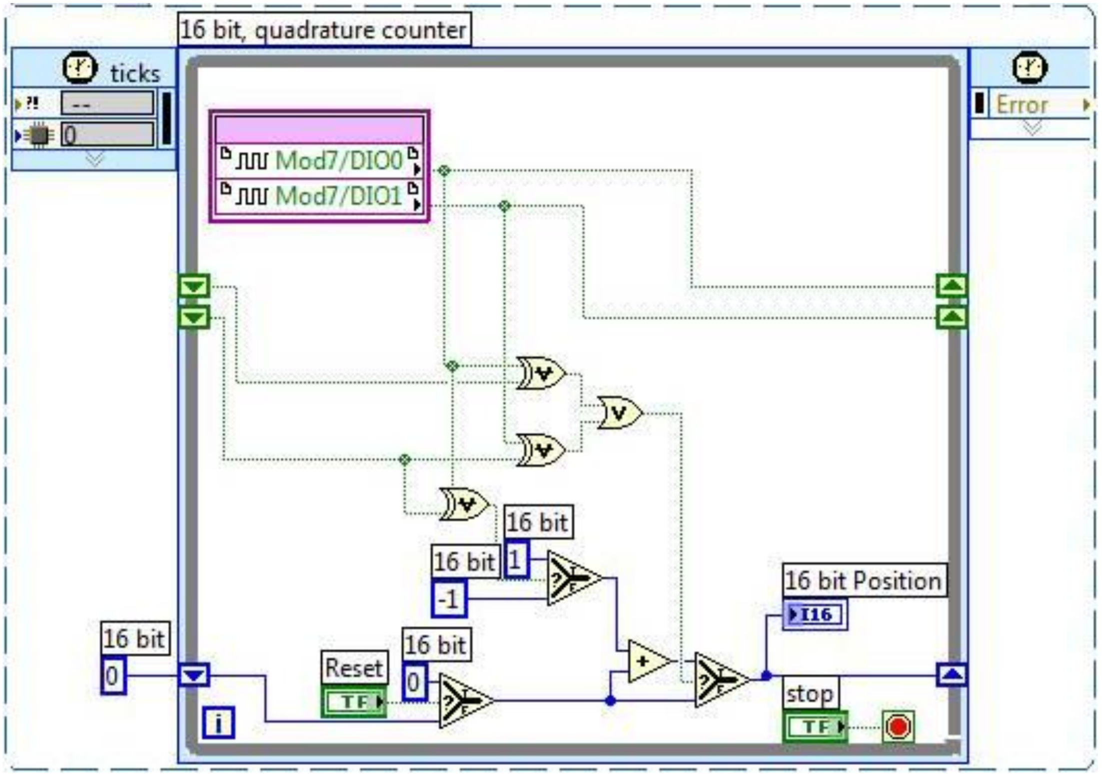
- Code is turned into “bitstream” of commands which configure the hardware (analogous to machine code on μ-processor)

The University of Nottingham

NI FPGA Compact RIO

FPGAs in practice





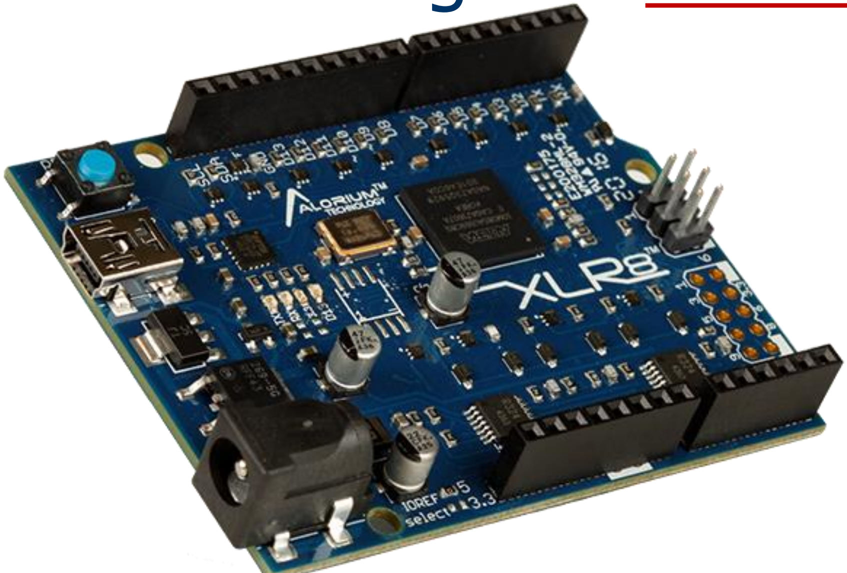
The University of Nottingham

Why bother with FPGAs?

- Better performance in some applications than microprocessors – hardware tends to be faster than software
- Much cheaper for moderate sized runs than application-specific ICs (ASICs)
- Truly parallel, no risk of time-critical tasks pre-empting one another
- Field-upgradable – no hardware redesign needed in case of modifications

An FPGA disguised as an Arduino!

- Example: Alorium XLR8 board is an Arduino Uno compatible board which has an FPGA instead of an Atmega microcontroller. It can be programmed using the Arduino IDE!



https://www.aloriumtech.com/xlr8/

Can you think of something we might have wanted to use an FPGA for?

- Something that needed to be done quicker than we could do it in software…
- We used dedicated hardware for it, but we could have done the same job via an FPGA
- What was it?

Summary

- Examined some issues of timing and scheduling of tasks including multitasking and multithreading
- Introduced polling for events
- Introduced interrupts and explored some applications you have previously seen
- Introduced FPGAs and explored how they work including the use of lookup tables
- Advantages and limitations of FPGAs

Can you please complete the module SEM survey?!