Lecture 03
Digital I/O and Interrupts

CE346 – Microprocessor System Design

Branden Ghena – Spring 2025

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley)

Northwestern

Administrivia

- Lab on Friday!
  - Frances Searle room 2370
  - See you all there!!
  - Show up on-time. We are going to get started right away
    - None of this 5-10 minutes late nonsense
- <u>Bring a USB-C adapter if you need one</u>
- Remember that you need to attend the section you registered for in CAESAR
  - If you need to switch for a day, ask me for permission in advance on Piazza

Quiz coming soon

- First quiz is next week Tuesday! (April 15)
  - 15-minute quiz, taken in-class on paper
  - Last fifteen minutes of class
  - Bring a pencil
  - No notes, no calculator

- Covers material from the last two weeks, including today

- Goals:
  - The quiz is not meant to be difficult. It’s meant to keep you involved
  - Do review class material and make sure you actually understand it

Today's Goals

- Two major parts today:

- 1. How does a microcontroller interact with peripherals?
  - Already discussed Memory-Mapped I/O
  - Interrupt and DMA mechanisms today

- 2. How do we interact with digital inputs and outputs?
  - GPIO peripheral for Digital I/O
  - GPIOTE peripheral for interrupts

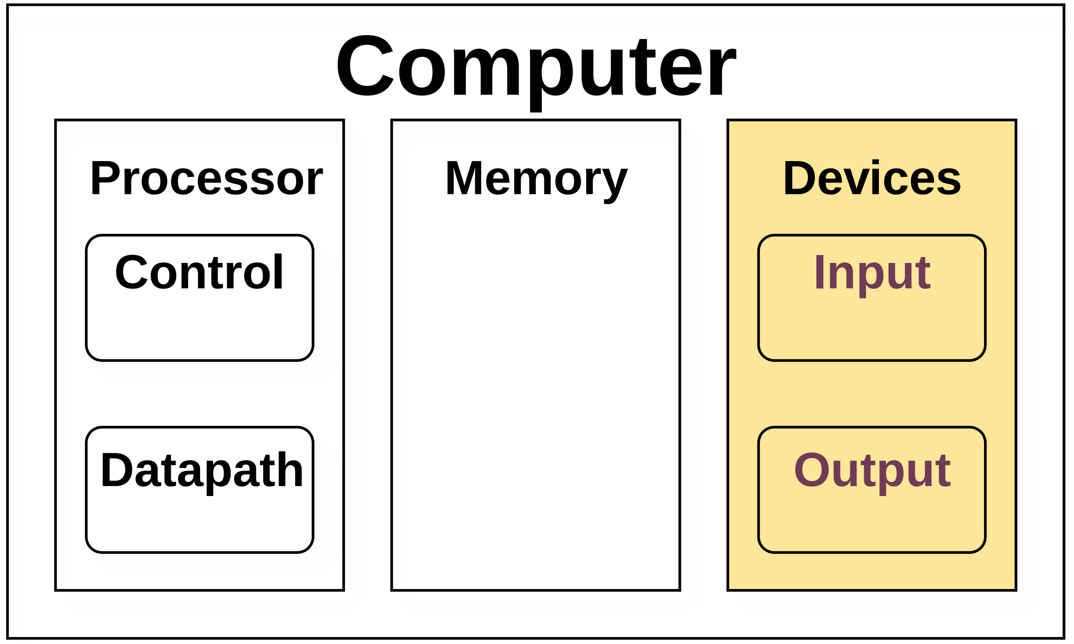
Outline

- **I/O Motivation**
- Pins
- Controlling digital signals: GIPO peripheral
- Interrupts
- Digital Input Interrupts: GPIOTE peripheral
- DMA (Direct Memory Access)

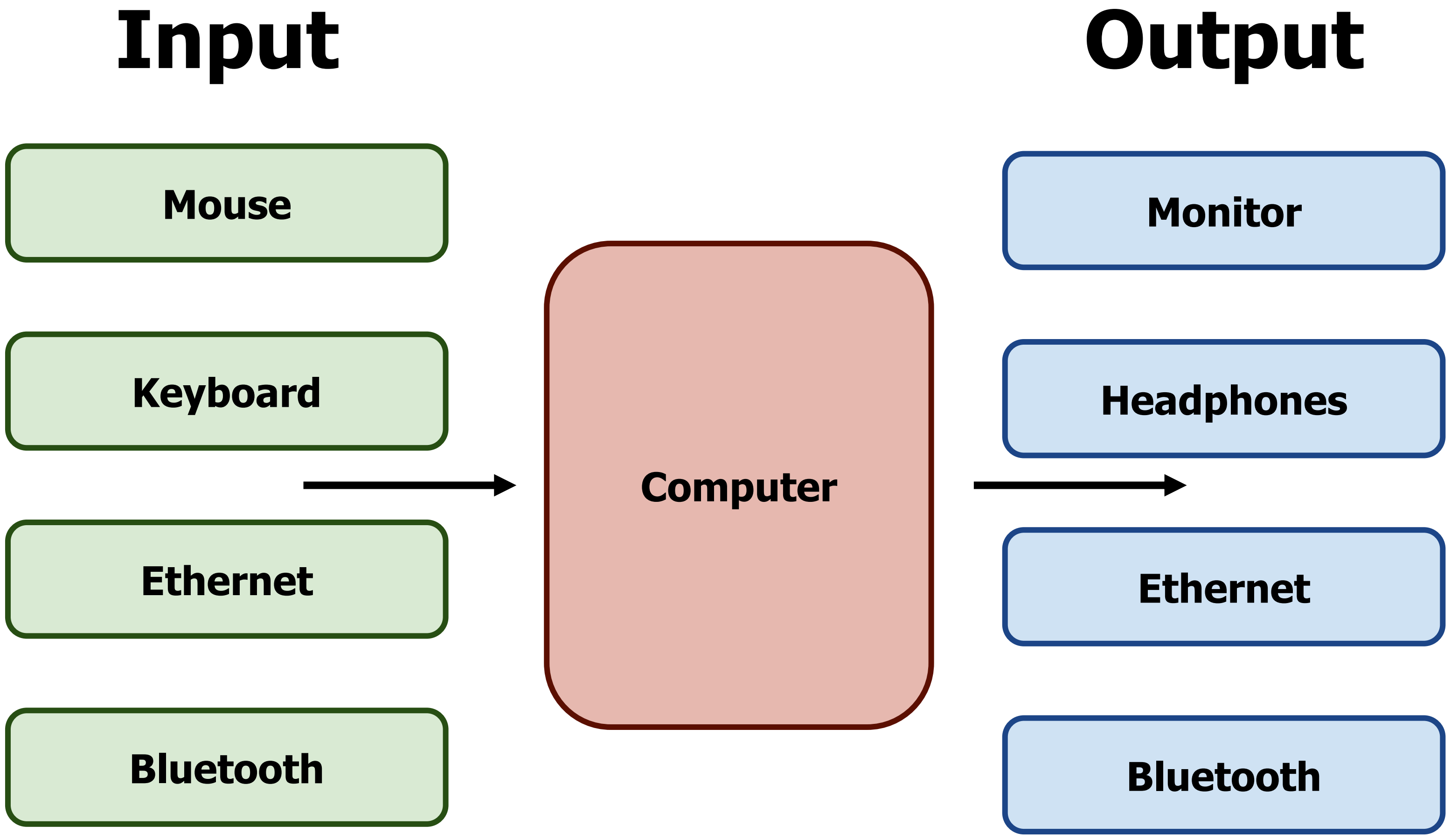
Devices are the point of computers

- Traditional systems need to receive input from users and output responses
  - Keyboard/mouse
  - Disk
  - Network
  - Graphics
  - Audio
  - Various USB devices

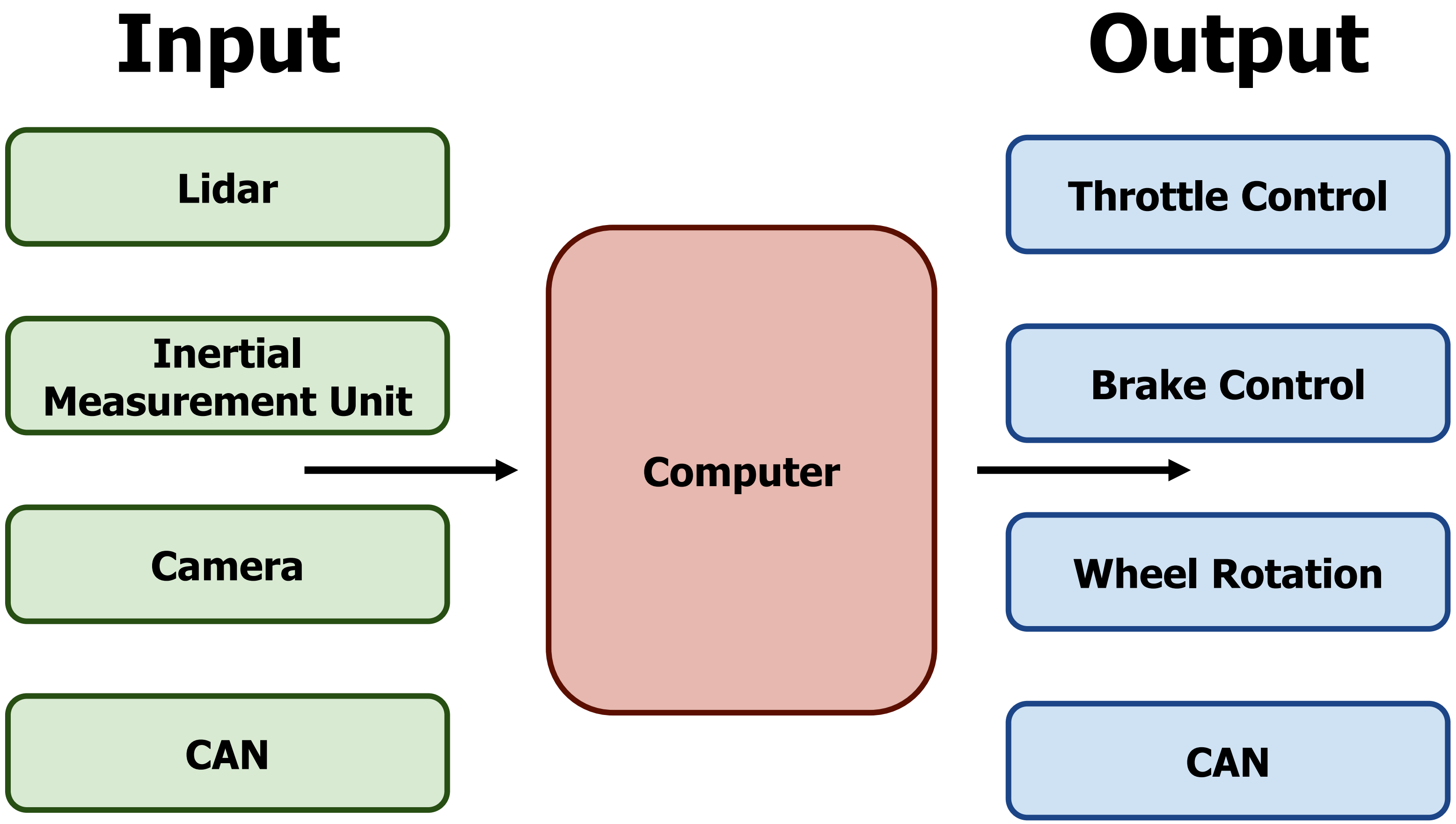
- Embedded systems have the same requirement, just more types of IO



Devices are core to useful general-purpose computing



Devices are essential to cyber-physical systems too



Device access rates vary by many orders of magnitude

- Rates in bit/sec
- System must be able to handle each of these
  - Sometimes needs low overhead
  - Sometimes needs to not wait around

Device	Behavior	Partner	Data Rate (Kb/s)
Keyboard	Input	Human	0.2
Mouse	Input	Human	0.4
Microphone	Output	Human	700.0
Bluetooth	Input or Output	Machine	20,000.0
Hard disk drive	Storage	Machine	100,000.0
Wireless network	Input or Output	Machine	300,000.0
Solid state drive	Storage	Machine	500,000.0
Wired LAN network	Input or Output	Machine	1,000,000.0
Graphics display	Output	Human	3,000,000.0

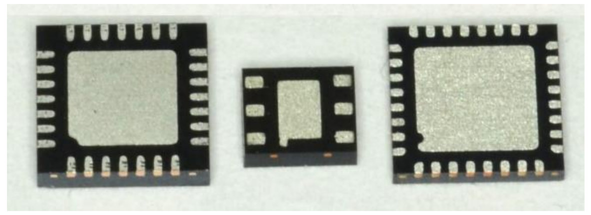
Outline

- I/O Motivation
- **Pins**
- Controlling digital signals: GIPO peripheral
- Interrupts
- Digital Input Interrupts: GPIOTE peripheral
- DMA (Direct Memory Access)

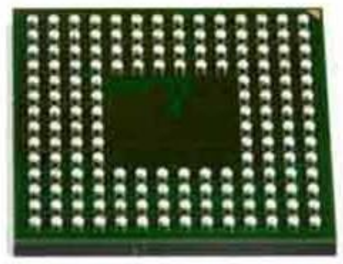
Pins

- Peripherals need to electrically connect to outside world
  - Attach to pins on the exterior of the chip

- More pins allow for more connections
  - At increased cost and size



- Under-chip pins can add more pins for cheaper
  - But make soldering and debugging difficult

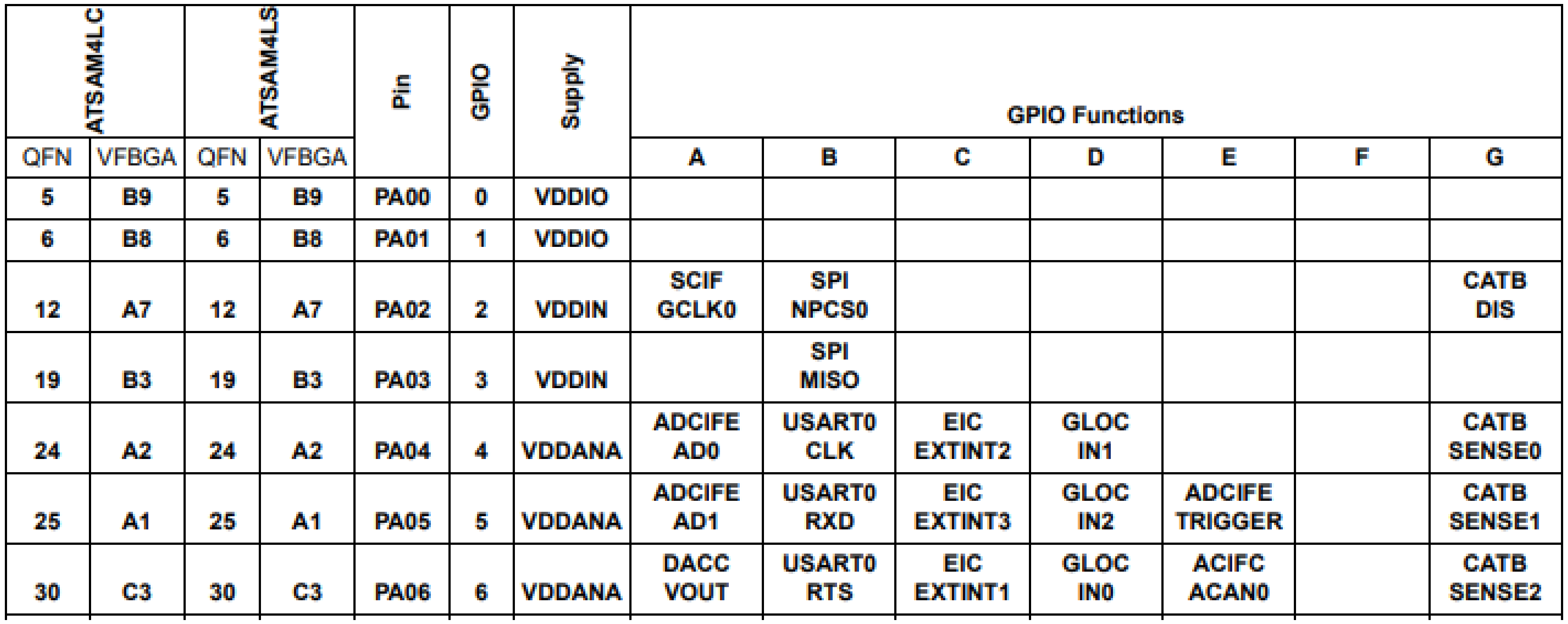


Internal connections to pins

- Peripherals need to connect to external pins
  - Can any peripheral connect to any pin, or are there limited mappings?
  - Modern microcontrollers allow any-to-any connections

- Older MCUs had mapping tables and pin selection was more challenging

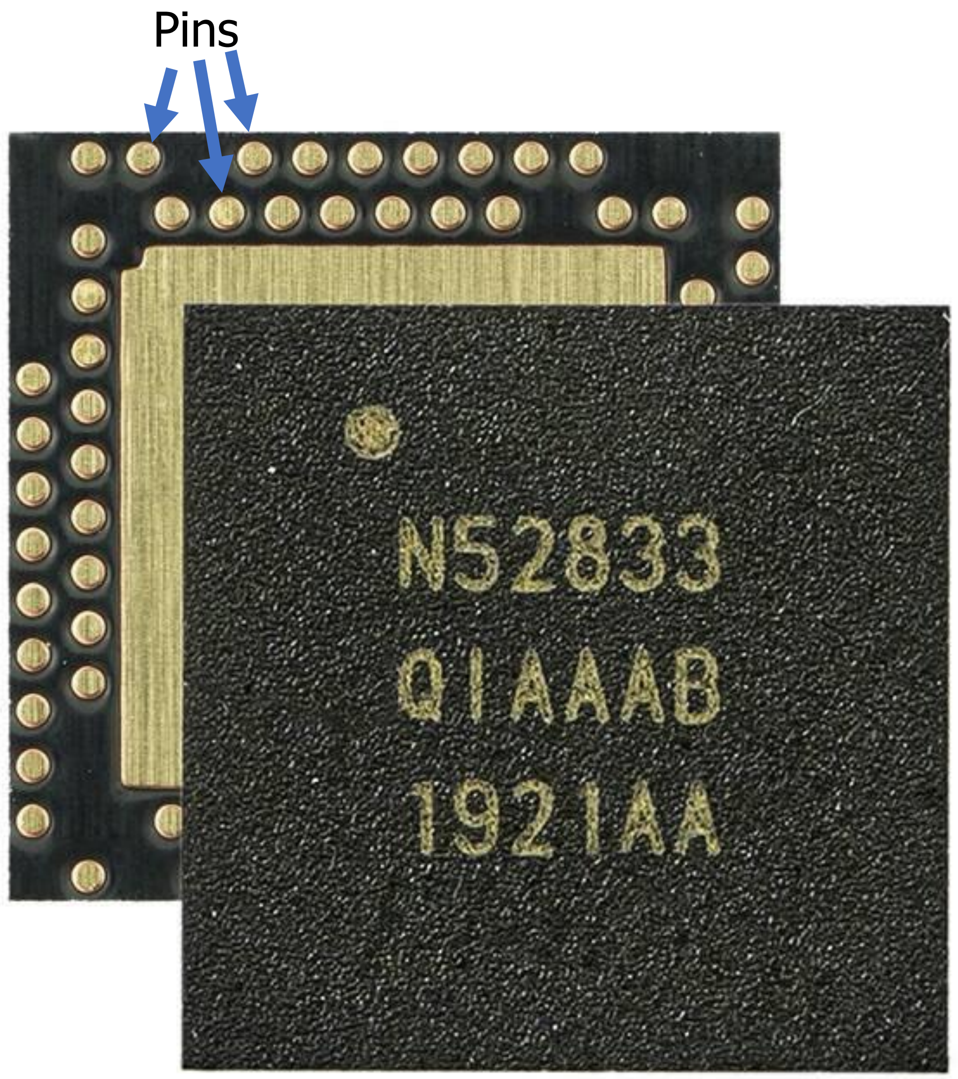
Table 3-1. 100-pin GPIO Controller Function Multiplexing (Sheet 1 of 4)



ATSAM4LC		ATSAM4LS		Pin	GPIO	Supply	GPIO Functions						
QFN	VFBGA	QFN	VFBGA				A	B	C	D	E	F	G
5	B9	5	B9	PA00	0	VDDIO							
6	B8	6	B8	PA01	1	VDDIO							
12	A7	12	A7	PA02	2	VDDIN	SCIF GCLK0	SPI NPCS0					CATB DIS
19	B3	19	B3	PA03	3	VDDIN		SPI MISO					
24	A2	24	A2	PA04	4	VDDANA	ADCIFE AD0	USART0 CLK	EIC EXTINT2	GLOC IN1			CATB SENSE0
25	A1	25	A1	PA05	5	VDDANA	ADCIFE AD1	USART0 RXD	EIC EXTINT3	GLOC IN2	ADCIFE TRIGGER		CATB SENSE1
30	C3	30	C3	PA06	6	VDDANA	DACC VOUT	USART0 RTS	EIC EXTINT1	GLOC IN0	ACIFC ACAN0		CATB SENSE2

Pins on the nRF52833

- 73 individual pins
  - Plus a big ground pad
- Some are used for special purposes
  - Power (16)
  - External Oscillator (2)
  - Debugging (2)
  - RF Antenna (1)
  - USB (2)
  - Not connected at all (8)
- Remaining 42 pins can be used as General Purpose I/O (GPIO)
  - Digital I/O
  - Or connected to other peripherals



Accessing nRF52833 pins in software

- Pins are collected into “Ports”
  - Each Port has 32 pins

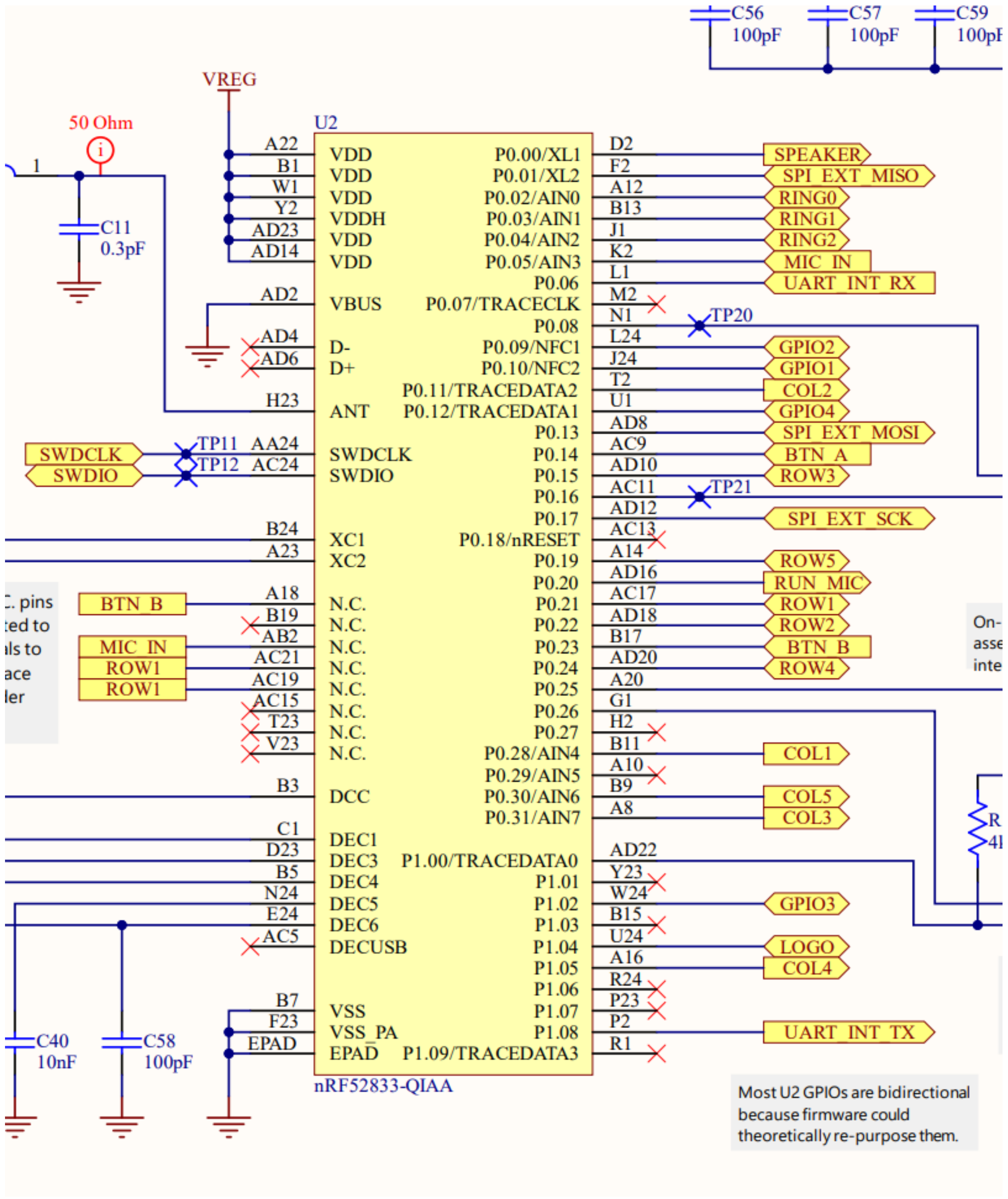
- 42 total GPIO pins
  - Port 0 – pins 0-31
  - Port 1 – pins 0-9

- Referred to as P`[port].[pin]`
  - For example: P0.20, P1.08, P0.00

Pins on the Microbit

- The Microbit uses the nRF52833, and then connects individual pins from the Microcontroller to certain parts of the board
  - Every board does this
  - The only way to know this connections is to look at the documentation

- For example:
  - BTN_A connects to P0.14
  - BTN_B connects to P0.23

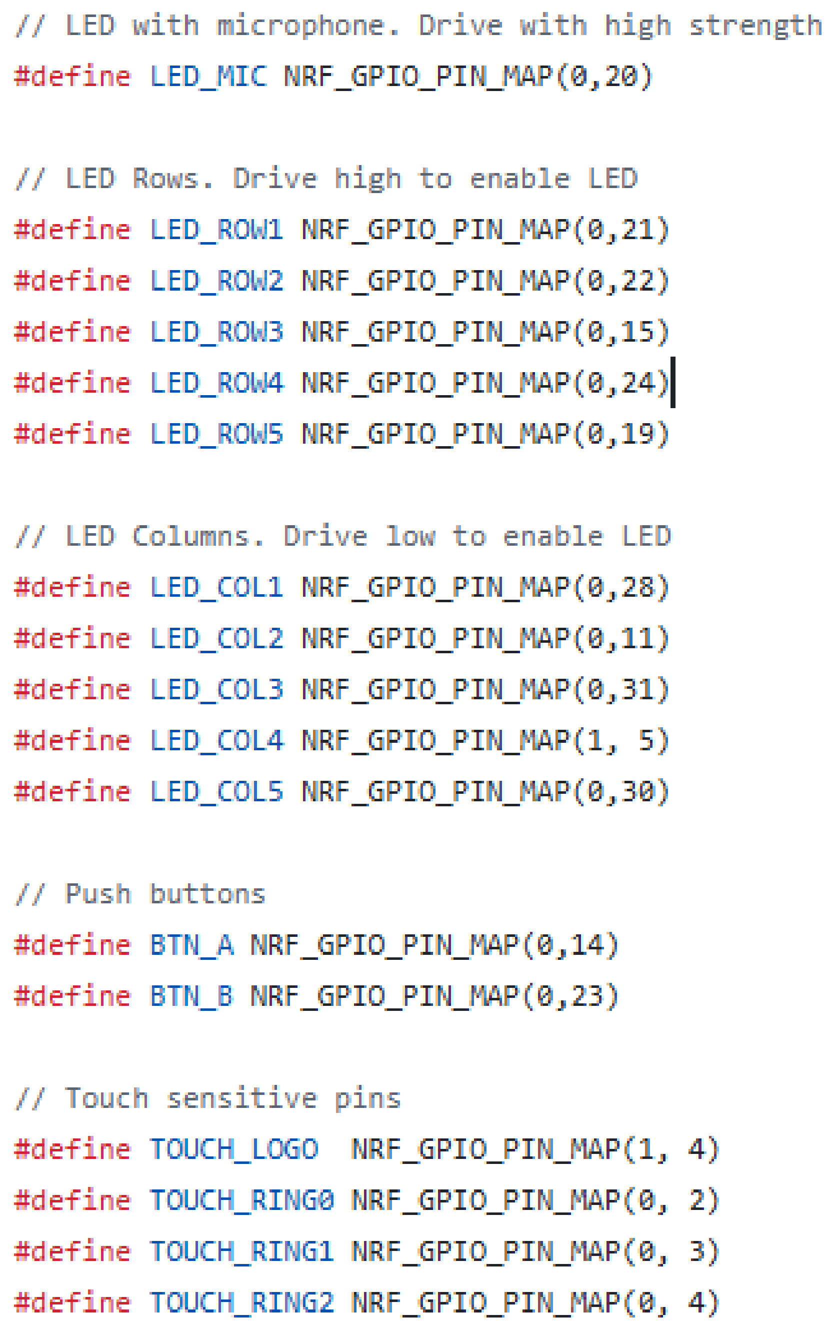


Most U2 GPIOs are bidirectional because firmware could theoretically re-purpose them.

In our software library

- Header file gives names to each pin based on how the Microbit uses it
  - https://github.com/nu-ce346/nu-microbit-base/blob/main/software/boards/microbit_v2/microbit_v2.h

- You can use these names directly in your code where you need the pin number



```
// LED with microphone. Drive with high strength
#define LED_MIC NRF_GPIO_PIN_MAP(0,20)

// LED Rows. Drive high to enable LED
#define LED_ROW1 NRF_GPIO_PIN_MAP(0,21)
#define LED_ROW2 NRF_GPIO_PIN_MAP(0,22)
#define LED_ROW3 NRF_GPIO_PIN_MAP(0,15)
#define LED_ROW4 NRF_GPIO_PIN_MAP(0,24)
#define LED_ROW5 NRF_GPIO_PIN_MAP(0,19)

// LED Columns. Drive low to enable LED
#define LED_COL1 NRF_GPIO_PIN_MAP(0,28)
#define LED_COL2 NRF_GPIO_PIN_MAP(0,11)
#define LED_COL3 NRF_GPIO_PIN_MAP(0,31)
#define LED_COL4 NRF_GPIO_PIN_MAP(1, 5)
#define LED_COL5 NRF_GPIO_PIN_MAP(0,30)

// Push buttons
#define BTN_A NRF_GPIO_PIN_MAP(0,14)
#define BTN_B NRF_GPIO_PIN_MAP(0,23)

// Touch sensitive pins
#define TOUCH_LOGO  NRF_GPIO_PIN_MAP(1, 4)
#define TOUCH_RING0 NRF_GPIO_PIN_MAP(0, 2)
#define TOUCH_RING1 NRF_GPIO_PIN_MAP(0, 3)
#define TOUCH_RING2 NRF_GPIO_PIN_MAP(0, 4)
```

Outline

- I/O Motivation
- Pins
- **Controlling digital signals: GIPO peripheral**
- Interrupts
- Digital Input Interrupts: GPIOTE peripheral
- DMA (Direct Memory Access)

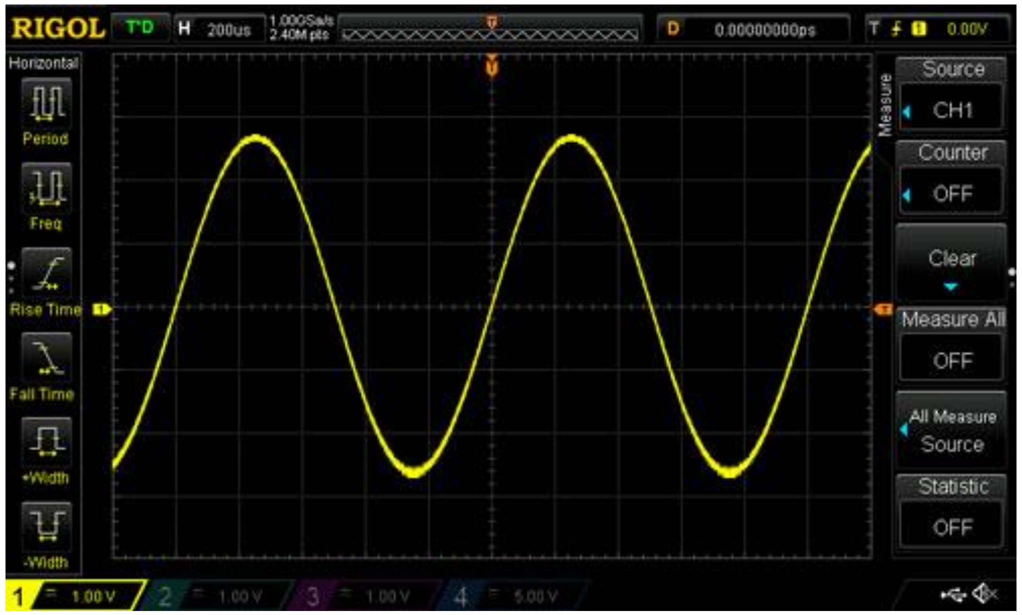
Digital signals

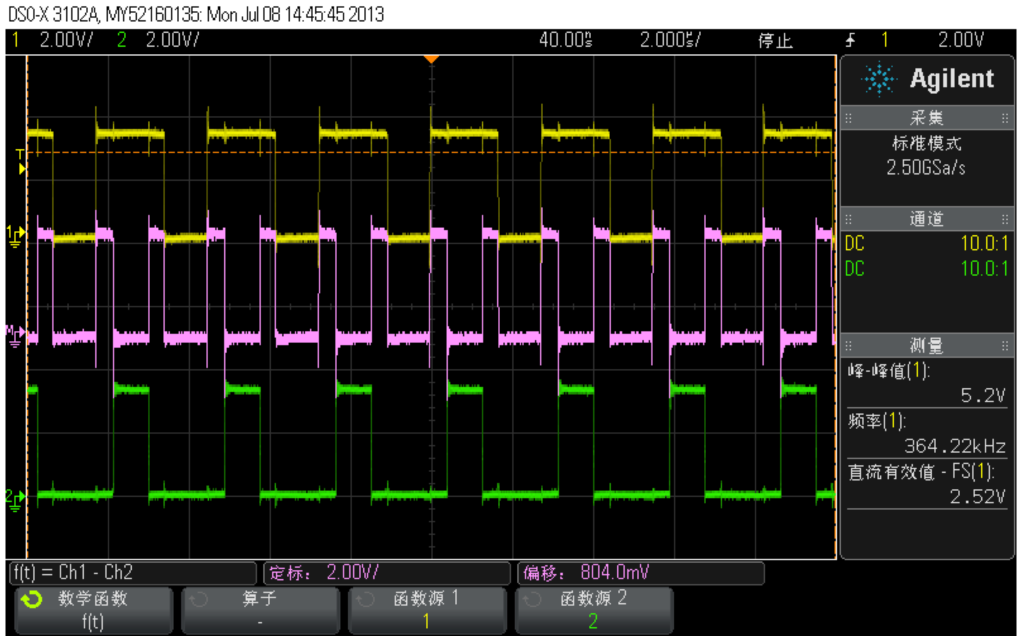
- Simplest form of I/O

- Exist in two states:
  - High (a.k.a. Set, a.k.a. 1)
  - Low (a.k.a. Clear, a.k.a. 0)

- Simpler to interact with
  - Constrained to two voltages
  - With quick transitions between the two

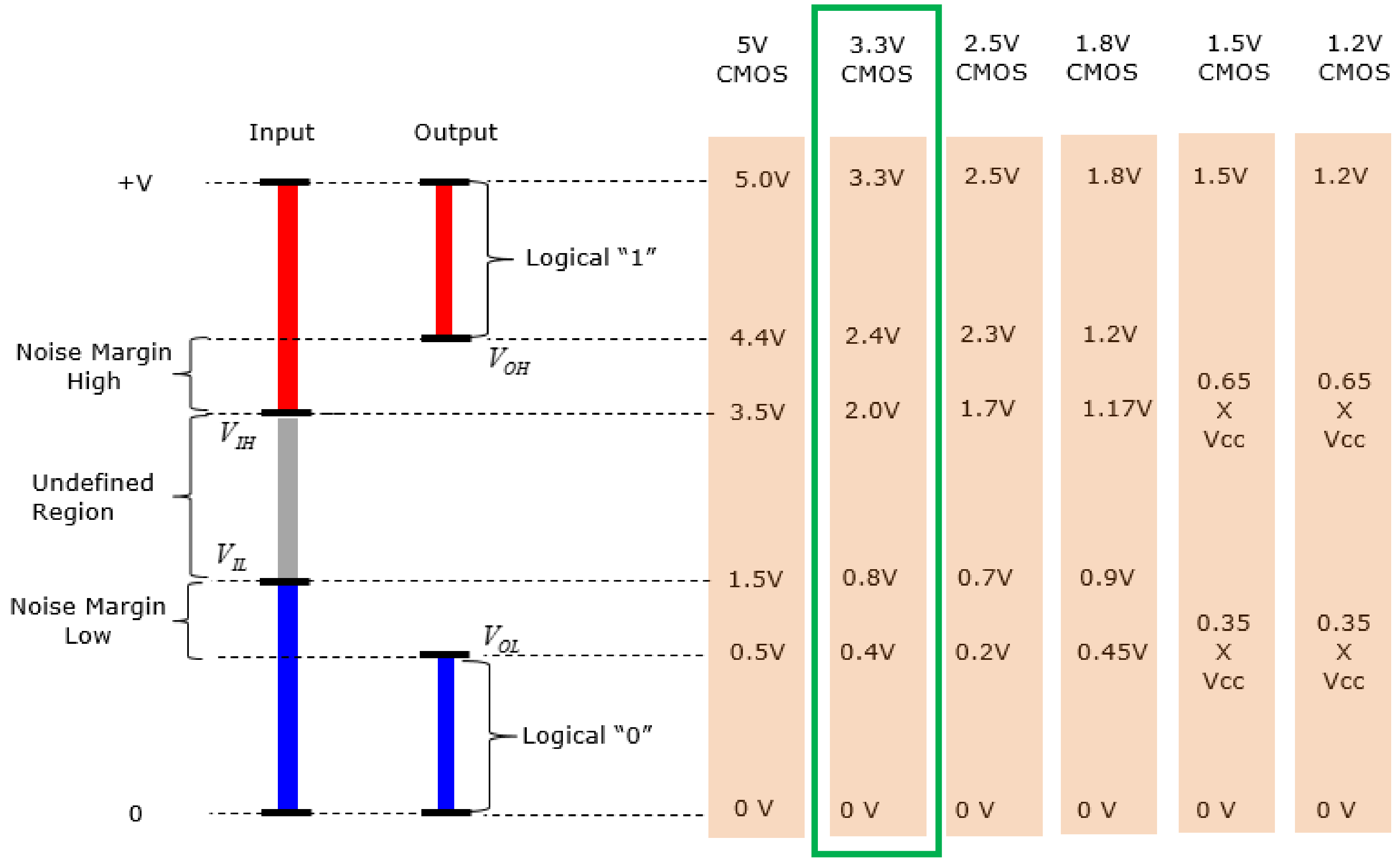
  - No math for voltage level
    - Either high or low





Digital signals map to voltage ranges

- Upper range is high signal
  - ~0.7*VDD
- Bottom range is low signal
  - ~0.3*VDD
- Middle is undefined
  - Only exists during transitions



	5V CMOS	3.3V CMOS	2.5V CMOS	1.8V CMOS	1.5V CMOS	1.2V CMOS
+V	5.0V	3.3V	2.5V	1.8V	1.5V	1.2V
V_{OH}	4.4V	2.4V	2.3V	1.2V		
V_{IH}	3.5V	2.0V	1.7V	1.17V	0.65 X Vcc	0.65 X Vcc
V_{IL}	1.5V	0.8V	0.7V	0.9V		
V_{OL}	0.5V	0.4V	0.2V	0.45V	0.35 X Vcc	0.35 X Vcc
0	0 V	0 V	0 V	0 V	0 V	0 V

http://www.sharetechnote.com/html/Electronics_CMOS.html

Digital Input/Output

- Read/write from/to external pins on the microcontroller
  - Two possible values: high (1) or low (0)

- Basic unit of operation for microcontrollers
  - Allows them to interact with buttons and LEDs
  - Every microcontroller has Digital I/O

Some terminology

- Pins can be inputs or outputs

- Output pins can be:
  - High or Set or 1
  - Low or Cleared or 0

- Input pins can read an externally connected value
  - Which is translated into either High or Low
  - Middle remains undefined. It’ll be High or Low though

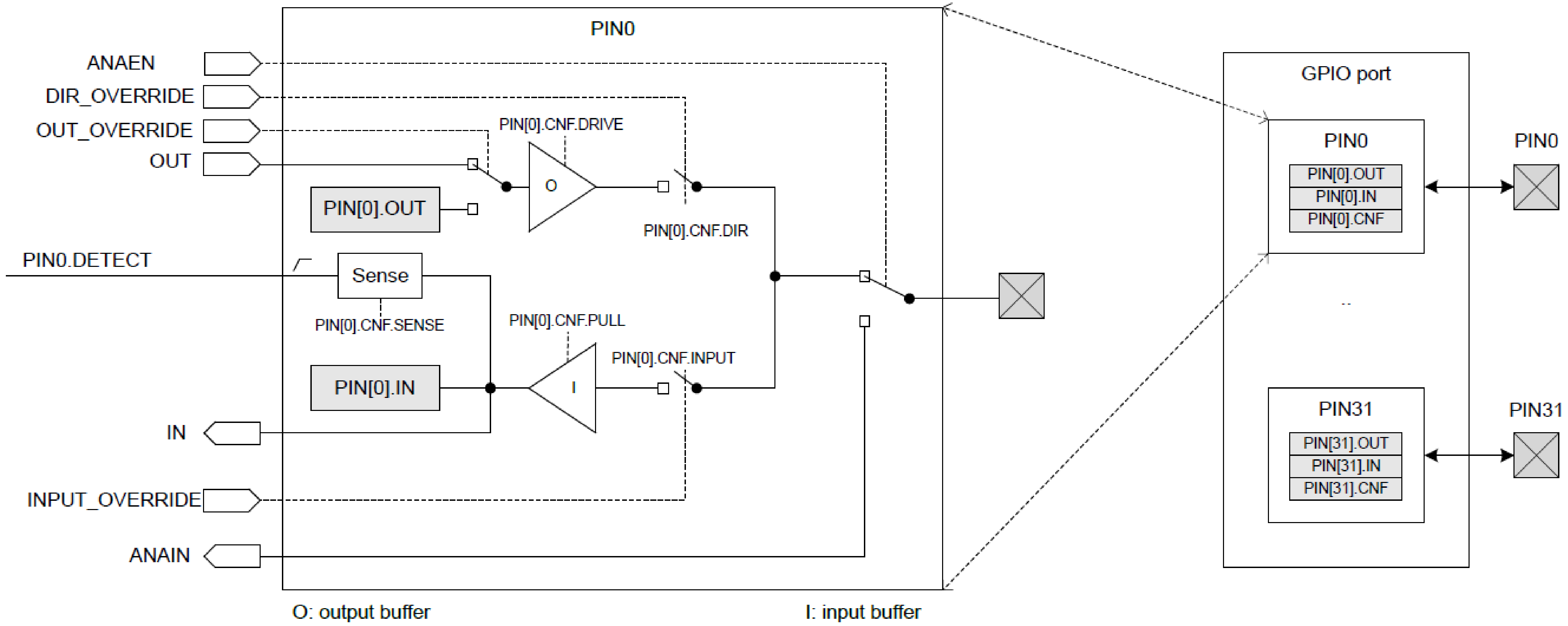
Controlling Digital I/O

- Outputs
  - Configure pin as an output
  - Set or Clear it

- Inputs
  - Configure pin as an input
  - Read it

- That’s about it. Digital I/O peripherals all look something like this
  - Every Digital I/O peripheral also has some “special features” of some kind or another, which vary with the Microcontroller

nRF52833 GPIO Peripheral



nRF52 GPIO Output

- Outputs a high or low signal

- Output configurations
  - High drive output (either for high, low, or both)
    - Sources or sinks additional current
      - For powering external devices
    - Normal drive: ~2 mA
    - High drive: ~10 mA

  - Disconnect (a.k.a. High Impedance or High-Z)
    - Wired-OR or Wired-AND scenarios (we’ll talk about these later in class)

nRF52 GPIO Input

- Reads in a signal as either high or low

- Input Configurations
  - Input buffer connect/disconnect
    - Allows the pin to be disabled if not being read from
  - Pull
    - Disabled, Pulldown, Pullup (we'll discuss in a future lecture)
    - Connects an internal pull up/down resistor (~13 kΩ)
    - Sets default value of input

Electrical specifications

- High voltage range: 0.7*VDD to VDD (~2.3 volts)
- Low voltage range: Ground to 0.3*VDD (~1 volt)

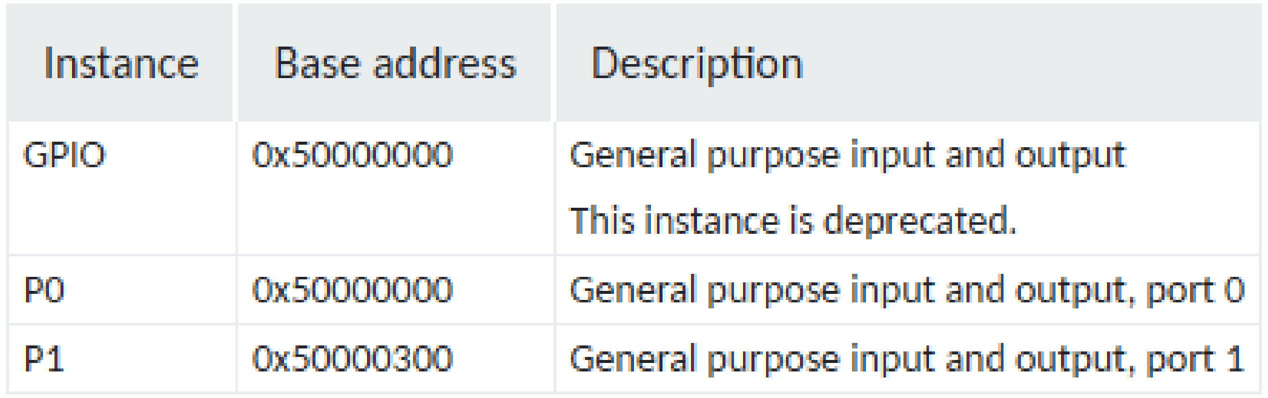
- GPIO are extremely fast
  - Transition time is <25 ns
  - Connected directly to memory bus for faster interactions

  - This allows complicated signal patterns to be replicated in software
    - If they aren’t implemented as a hardware peripheral
    - Known as “bit-banging”

GPIO Instances

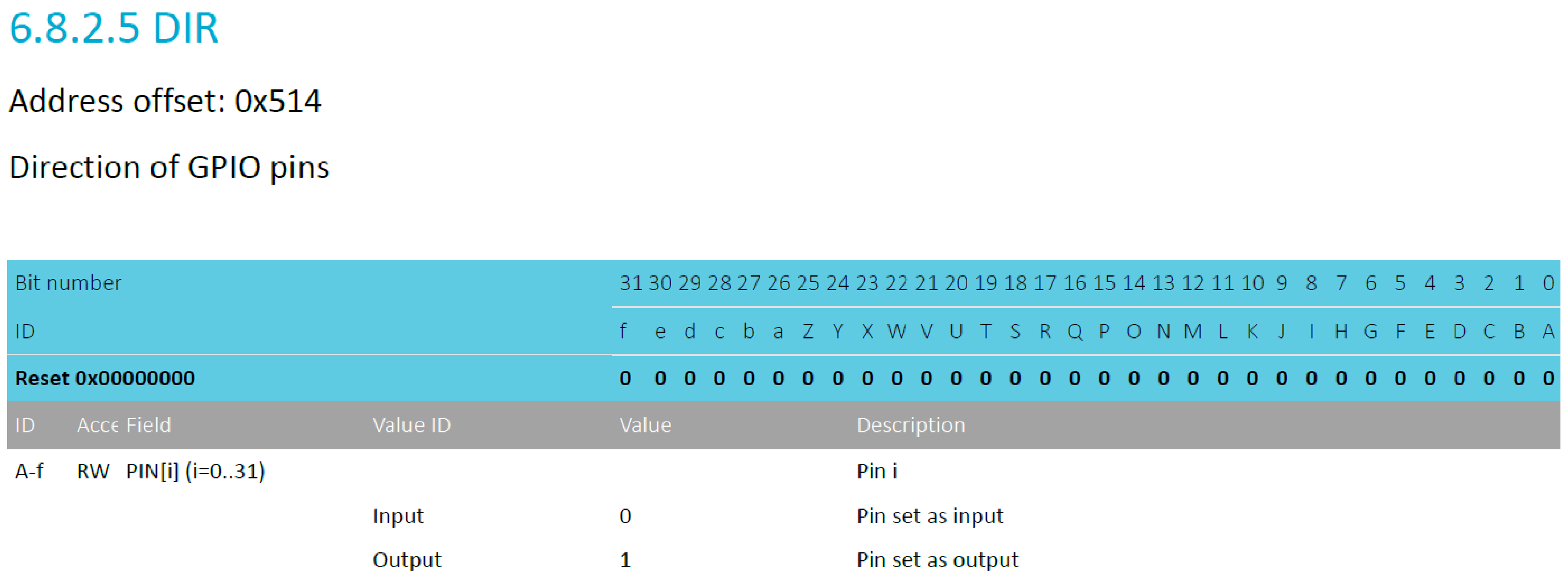
- The nRF52833 has two GPIO peripheral “instances”
  - Several peripherals have multiple instances. For example, multiple timers
  - Each instance functions independently
  - Each instance has its own copy of MMIO registers

- For the GPIO, each instance controls one Port
  - Port 0 with 32 pins
  - Port 1 with 10 pins



Instance	Base address	Description
GPIO	0x50000000	General purpose input and output This instance is deprecated.
P0	0x50000000	General purpose input and output, port 0
P1	0x50000300	General purpose input and output, port 1

Pin configuration



6.8.2.5 DIR

Address offset: 0x514

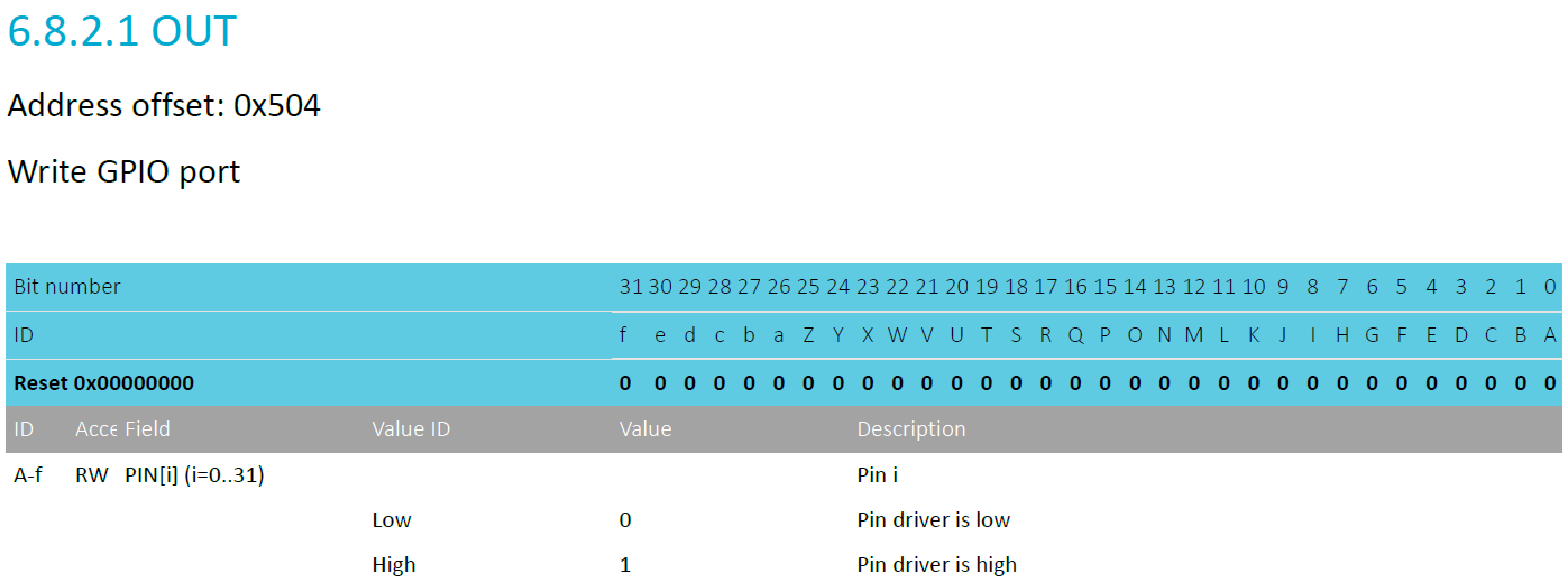
Direction of GPIO pins

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID	f	e	d	c	b	a	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset 0x00000000	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

ID	Acce	Field	Value ID	Value	Description
A-f	RW	PIN[i] (i=0..31)			Pin i
			Input	0	Pin set as input
			Output	1	Pin set as output

- DIR register controls direction (input or output) for each pin
  - Each bit 0-31 corresponds to pin 0-31
  - Reset value: 0x00000000 -> all pins are inputs by default

Controlling output level



6.8.2.1 OUT

Address offset: 0x504

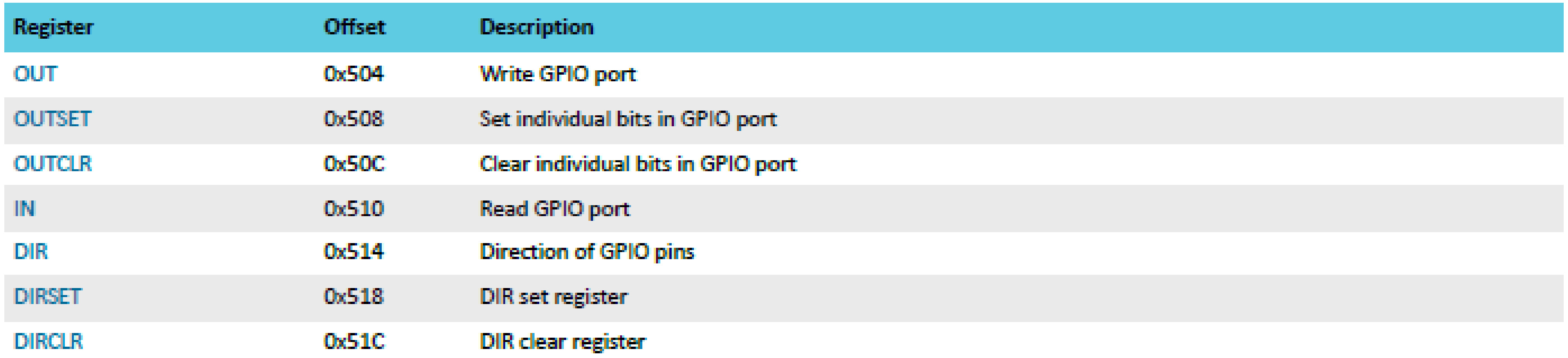
Write GPIO port

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID	f	e	d	c	b	a	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset 0x00000000	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

ID	Acce	Field	Value ID	Value	Description
A-f	RW	PIN[i] (i=0..31)			Pin i
			Low	0	Pin driver is low
			High	1	Pin driver is high

- OUT register controls whether each pin is high or low
  - Only meaningful if the pin is configured as an Output
  - Again, each bit is a single pin and reset is 0x00000000 (all pins low)

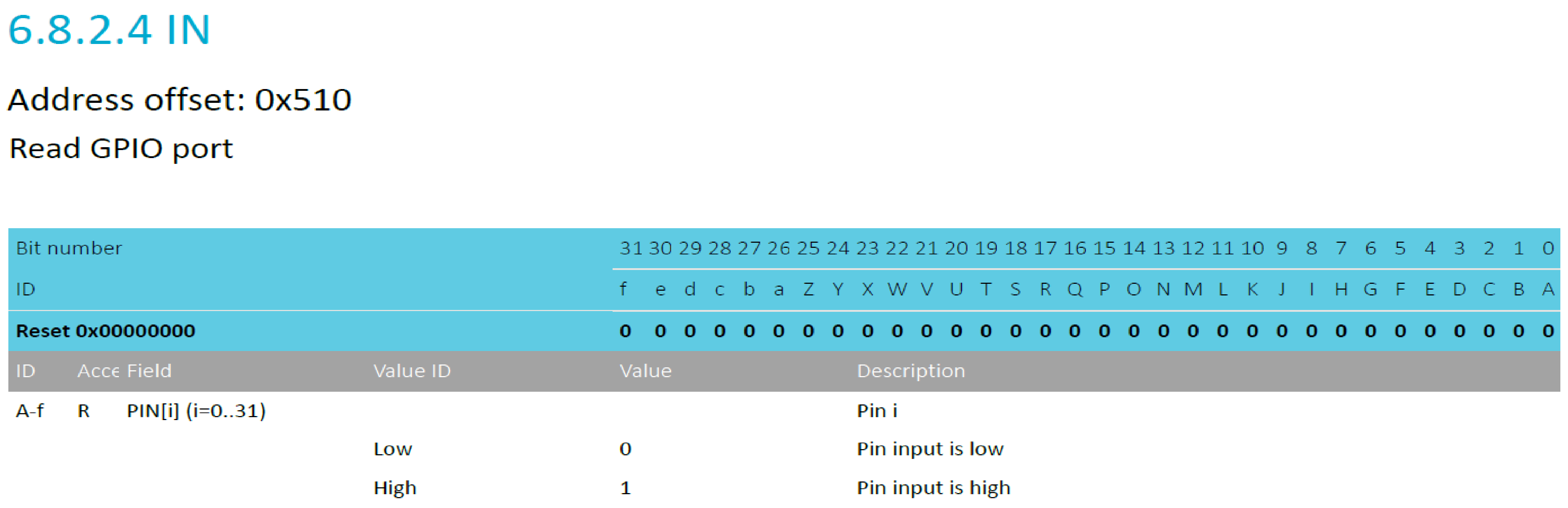
*Set/*Clear registers



Register	Offset	Description
OUT	0x504	Write GPIO port
OUTSET	0x508	Set individual bits in GPIO port
OUTCLR	0x50C	Clear individual bits in GPIO port
IN	0x510	Read GPIO port
DIR	0x514	Direction of GPIO pins
DIRSET	0x518	DIR set register
DIRCLR	0x51C	DIR clear register

- OUT works traditionally: write a 1 for high, 0 for low
- OUTSET write a 1 to set that pin (high) zero has no effect
- OUTCLR write a 1 to clear that pin (low) zero has no effect
  - Lets you modify a pin without modifying the others (or reading first)

Reading input levels



6.8.2.4 IN

Address offset: 0x510

Read GPIO port

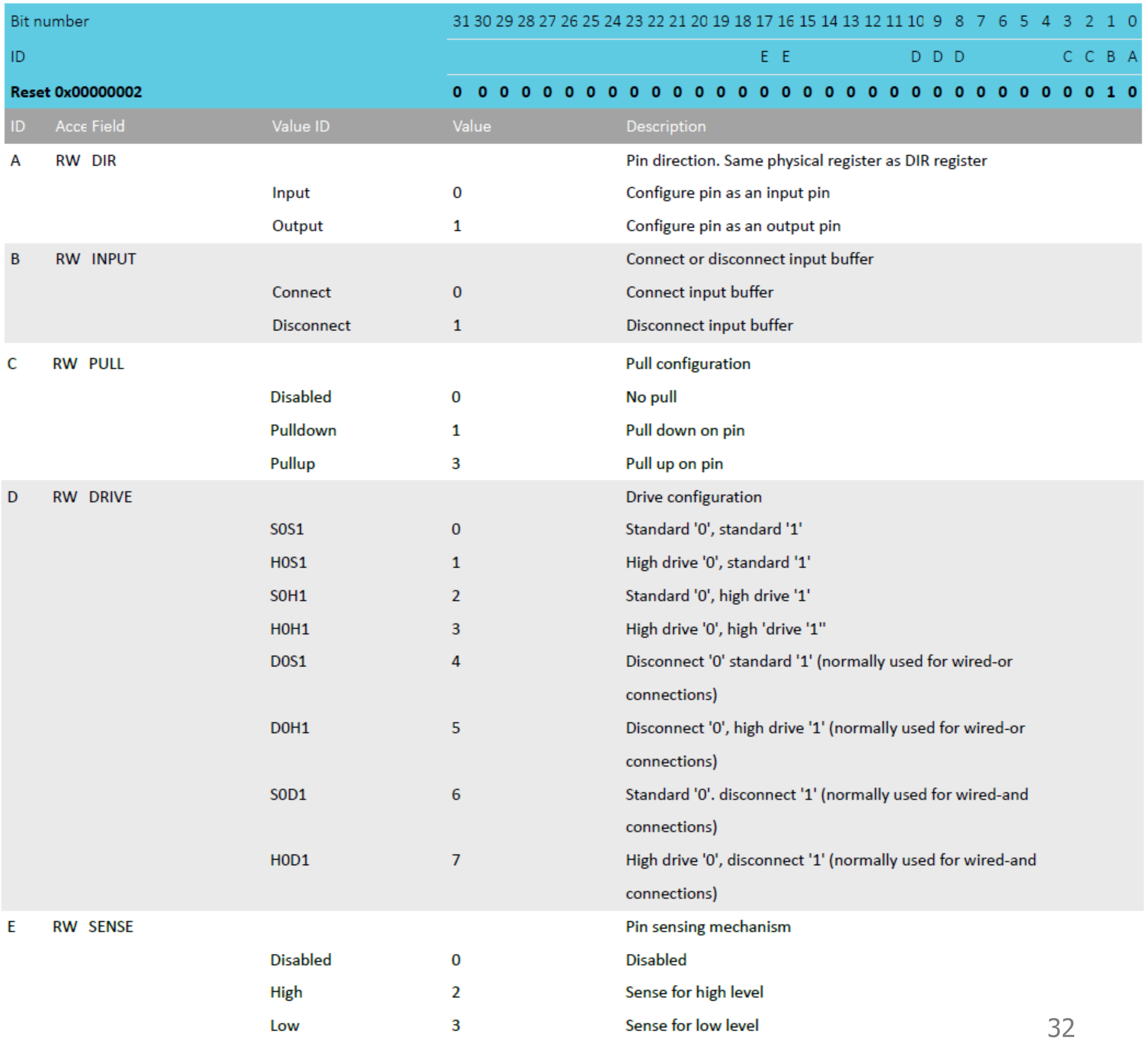
Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID	f	e	d	c	b	a	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset 0x00000000	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

ID	Acce	Field	Value ID	Value	Description
A-f	R	PIN[i] (i=0..31)			Pin i
			Low	0	Pin input is low
			High	1	Pin input is high

- IN register allows you to read the value for an input pin
  - Each bit 0-31 corresponds to pin 0-31
  - Read-only register. Writing has no effect
  - For Port1, pins >= 10 are *undefined* (data sheet doesn't mention it)

Complex configuration

- If you want to change other pin configurations, you do so per pin with the `PIN_CNF[n]` registers
    - There are 32 of them, one per pin
- Various fields correspond to different groups of bits
    - Direction, Input buffer, Pullup/down, Drive strength, Sensing mechanism
- Bits not part of a field should be ignored
    - Do not modify them



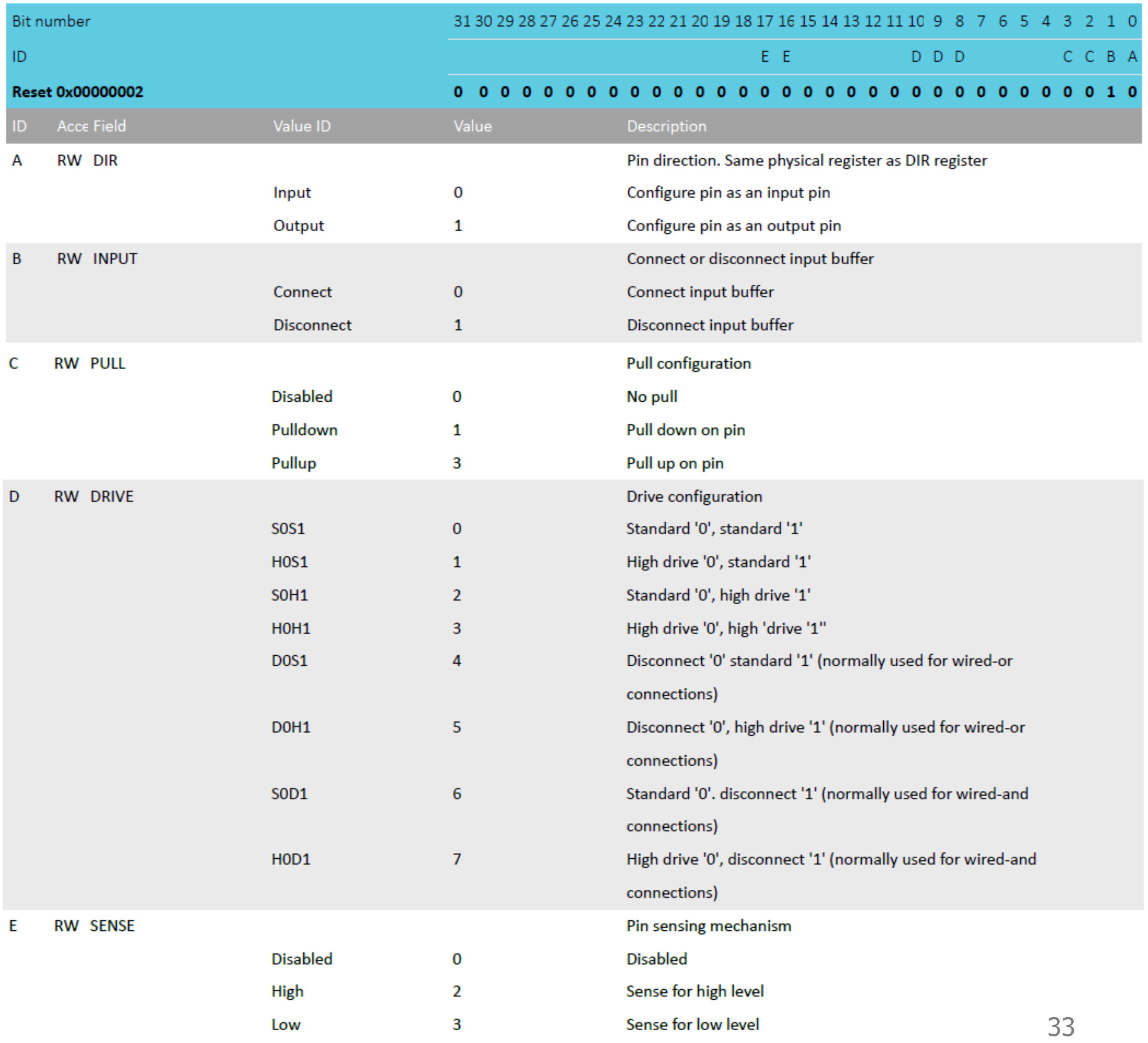
Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID															E	E						D	D	D					C	C	B	A
Reset 0x00000002	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

ID	Acce	Field	Value ID	Value	Description
A	RW	DIR			Pin direction. Same physical register as DIR register
			Input	0	Configure pin as an input pin
			Output	1	Configure pin as an output pin
B	RW	INPUT			Connect or disconnect input buffer
			Connect	0	Connect input buffer
			Disconnect	1	Disconnect input buffer
C	RW	PULL			Pull configuration
			Disabled	0	No pull
			Pulldown	1	Pull down on pin
			Pullup	3	Pull up on pin
D	RW	DRIVE			Drive configuration
			S0S1	0	Standard '0', standard '1'
			H0S1	1	High drive '0', standard '1'
			S0H1	2	Standard '0', high drive '1'
			H0H1	3	High drive '0', high 'drive '1''
			D0S1	4	Disconnect '0' standard '1' (normally used for wired-or connections)
			D0H1	5	Disconnect '0', high drive '1' (normally used for wired-or connections)
			S0D1	6	Standard '0'. disconnect '1' (normally used for wired-and connections)
			H0D1	7	High drive '0', disconnect '1' (normally used for wired-and connections)
E	RW	SENSE			Pin sensing mechanism
			Disabled	0	Disabled
			High	2	Sense for high level
			Low	3	Sense for low level

Break + Question

- Reset value is 0x2

- **What are the default configurations here?**



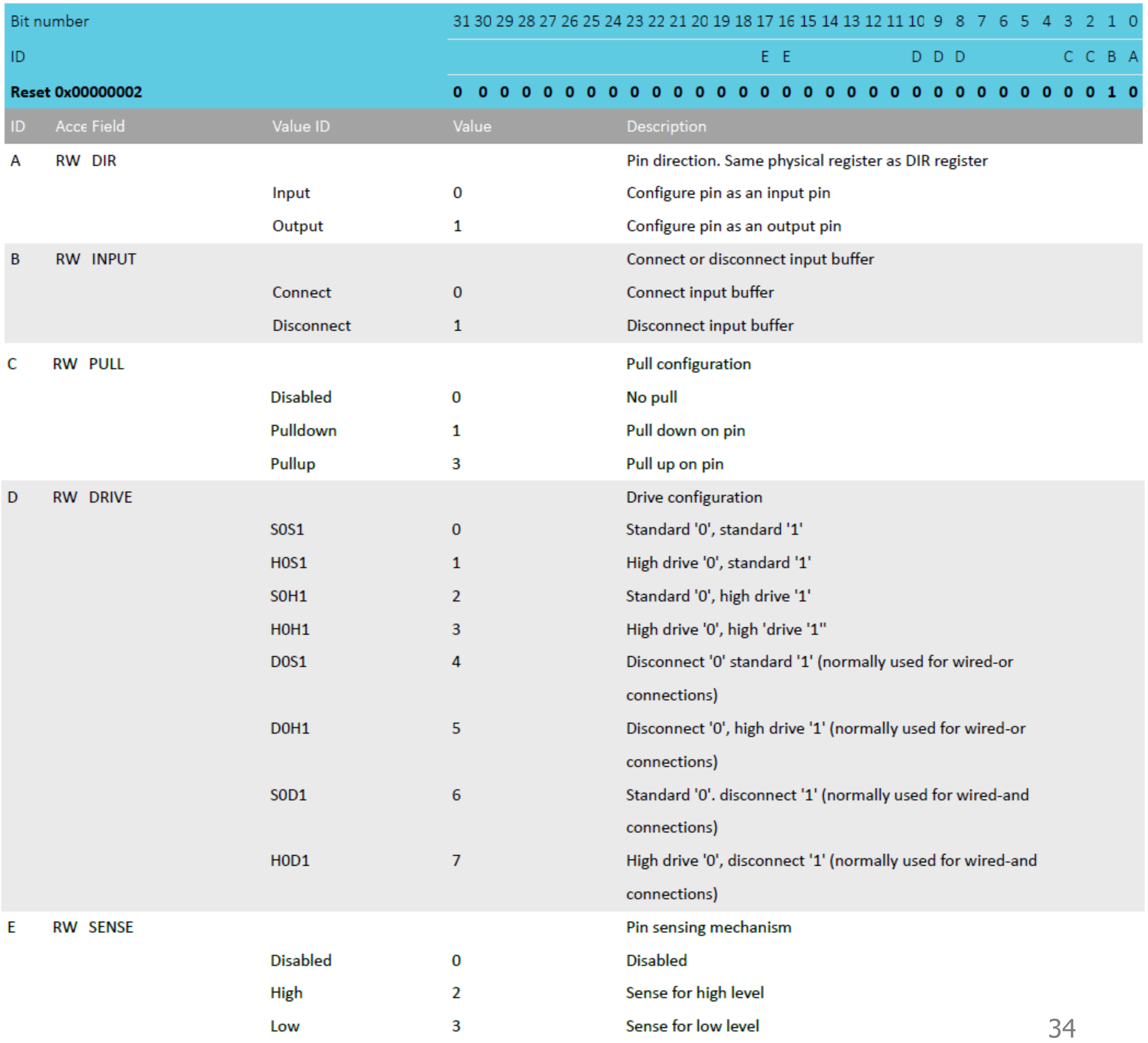
Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID															E	E						D	D	D					C	C	B	A
Reset 0x00000002	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

ID	Acce	Field	Value ID	Value	Description
A	RW	DIR			Pin direction. Same physical register as DIR register
			Input	0	Configure pin as an input pin
			Output	1	Configure pin as an output pin
B	RW	INPUT			Connect or disconnect input buffer
			Connect	0	Connect input buffer
			Disconnect	1	Disconnect input buffer
C	RW	PULL			Pull configuration
			Disabled	0	No pull
			Pulldown	1	Pull down on pin
			Pullup	3	Pull up on pin
D	RW	DRIVE			Drive configuration
			S0S1	0	Standard '0', standard '1'
			H0S1	1	High drive '0', standard '1'
			S0H1	2	Standard '0', high drive '1'
			H0H1	3	High drive '0', high 'drive '1''
			D0S1	4	Disconnect '0' standard '1' (normally used for wired-or connections)
			D0H1	5	Disconnect '0', high drive '1' (normally used for wired-or connections)
			S0D1	6	Standard '0'. disconnect '1' (normally used for wired-and connections)
			H0D1	7	High drive '0', disconnect '1' (normally used for wired-and connections)
E	RW	SENSE			Pin sensing mechanism
			Disabled	0	Disabled
			High	2	Sense for high level
			Low	3	Sense for low level

Break + Question

- Reset value is 0x2

- **What are the default configurations here?**
  - DIR: Input
  - INPUT: Disconnected
  - PULL: Disabled
  - DRIVE: Standard 0/1
  - SENSE: Disabled



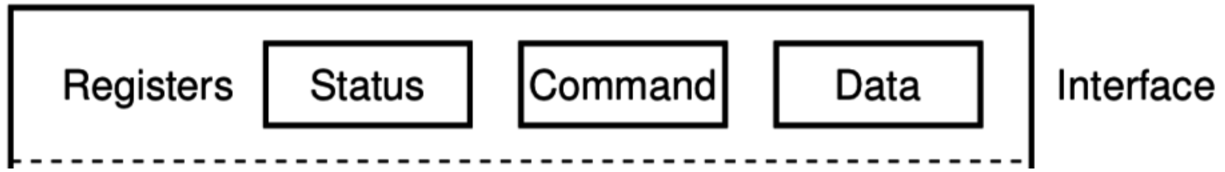
Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID															E	E						D	D	D					C	C	B	A
Reset 0x00000002	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

ID	Acce	Field	Value ID	Value	Description
A	RW	DIR			Pin direction. Same physical register as DIR register
			Input	0	Configure pin as an input pin
			Output	1	Configure pin as an output pin
B	RW	INPUT			Connect or disconnect input buffer
			Connect	0	Connect input buffer
			Disconnect	1	Disconnect input buffer
C	RW	PULL			Pull configuration
			Disabled	0	No pull
			Pulldown	1	Pull down on pin
			Pullup	3	Pull up on pin
D	RW	DRIVE			Drive configuration
			S0S1	0	Standard '0', standard '1'
			H0S1	1	High drive '0', standard '1'
			S0H1	2	Standard '0', high drive '1'
			H0H1	3	High drive '0', high 'drive '1''
			D0S1	4	Disconnect '0' standard '1' (normally used for wired-or connections)
			D0H1	5	Disconnect '0', high drive '1' (normally used for wired-or connections)
			S0D1	6	Standard '0'. disconnect '1' (normally used for wired-and connections)
			H0D1	7	High drive '0', disconnect '1' (normally used for wired-and connections)
E	RW	SENSE			Pin sensing mechanism
			Disabled	0	Disabled
			High	2	Sense for high level
			Low	3	Sense for low level

Outline

- I/O Motivation
- Pins
- Controlling digital signals: GIPO peripheral
- **Interrupts**
- Digital Input Interrupts: GPIOTE peripheral
- DMA (Direct Memory Access)

What do interactions with devices look like?



1. while STATUS==BUSY; Wait
   - (Need to make sure device is ready for a command)
2. Write value(s) to DATA
3. Write command(s) to COMMAND
4. while STATUS==BUSY; Wait
   - (Need to make sure device has completed the request)
5. Read value(s) from Data

This is the "polling" model of I/O.

"Poll" the peripheral in software repeatedly to see if it's ready yet.

Waiting can be a waste of CPU time

1. **while STATUS==BUSY; Wait**
   - **(Need to make sure device is ready for a command)**
2. Write value(s) to DATA
3. Write command(s) to COMMAND
4. **while STATUS==BUSY; Wait**
   - **(Need to make sure device has completed the request)**
5. Read value(s) from Data

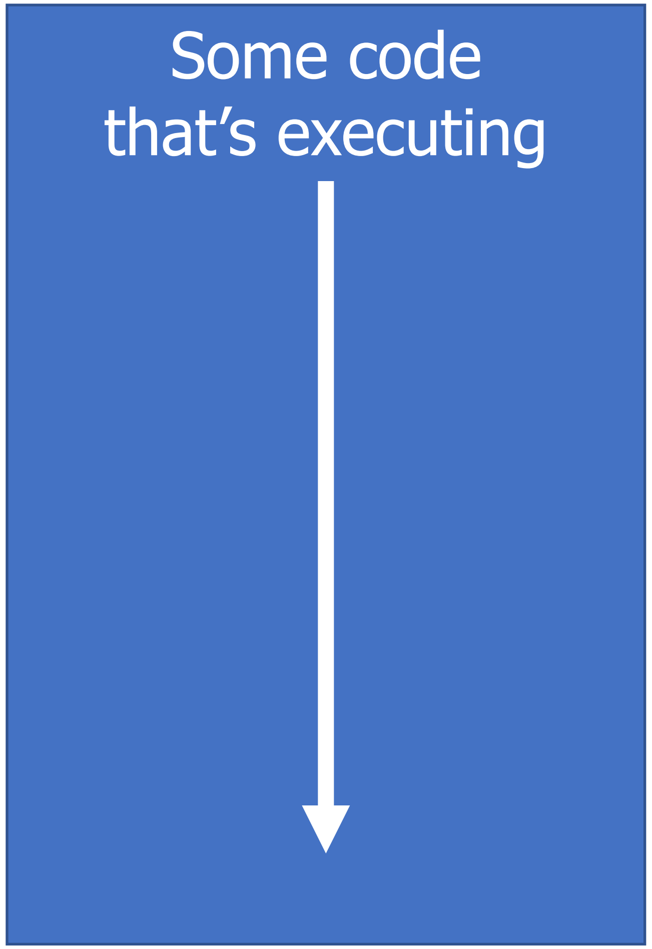
- Problem: imagine a keyboard device
  - CPU could be waiting for minutes before data arrives
  - Need a way to notify CPU when an event occurs
    - Interrupts!

Interrupts

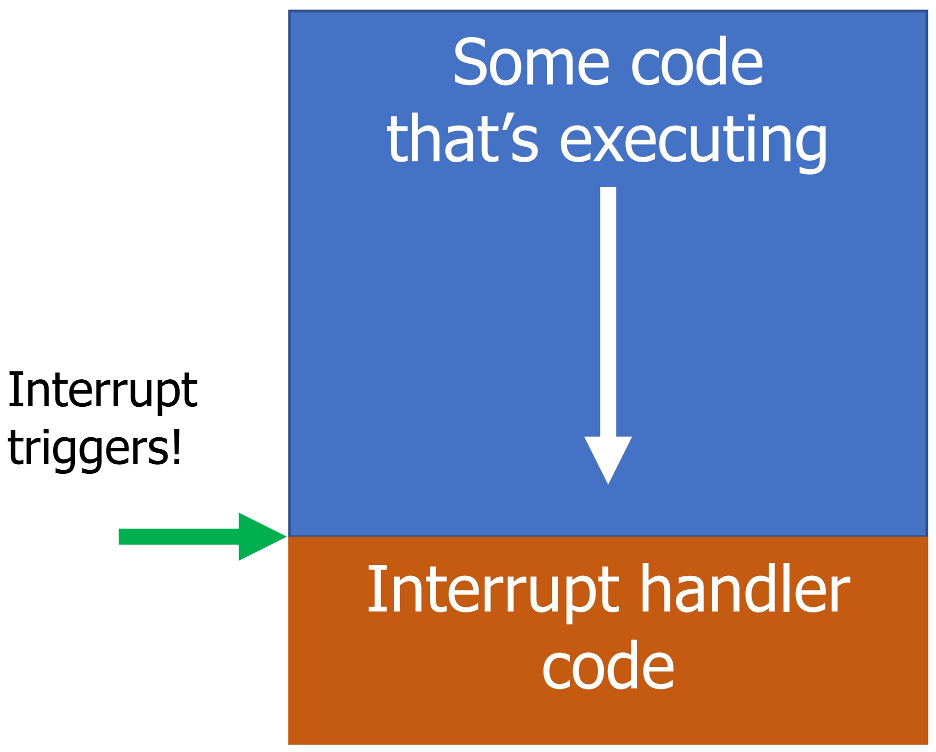
- What is an interrupt?
    - Some event which causes the processor to stop normal execution
    - The processor instead jumps to a software “handler” for that event
        - Then returns back to what it was doing afterwards

- What causes interrupts?
    - Hardware exceptions
        - Divide by zero, Undefined Instruction, Memory bus error
    - Software
        - Syscall, Software Interrupt (SWI)
    - External hardware
        - Input pin, Timer, various “Data Ready”

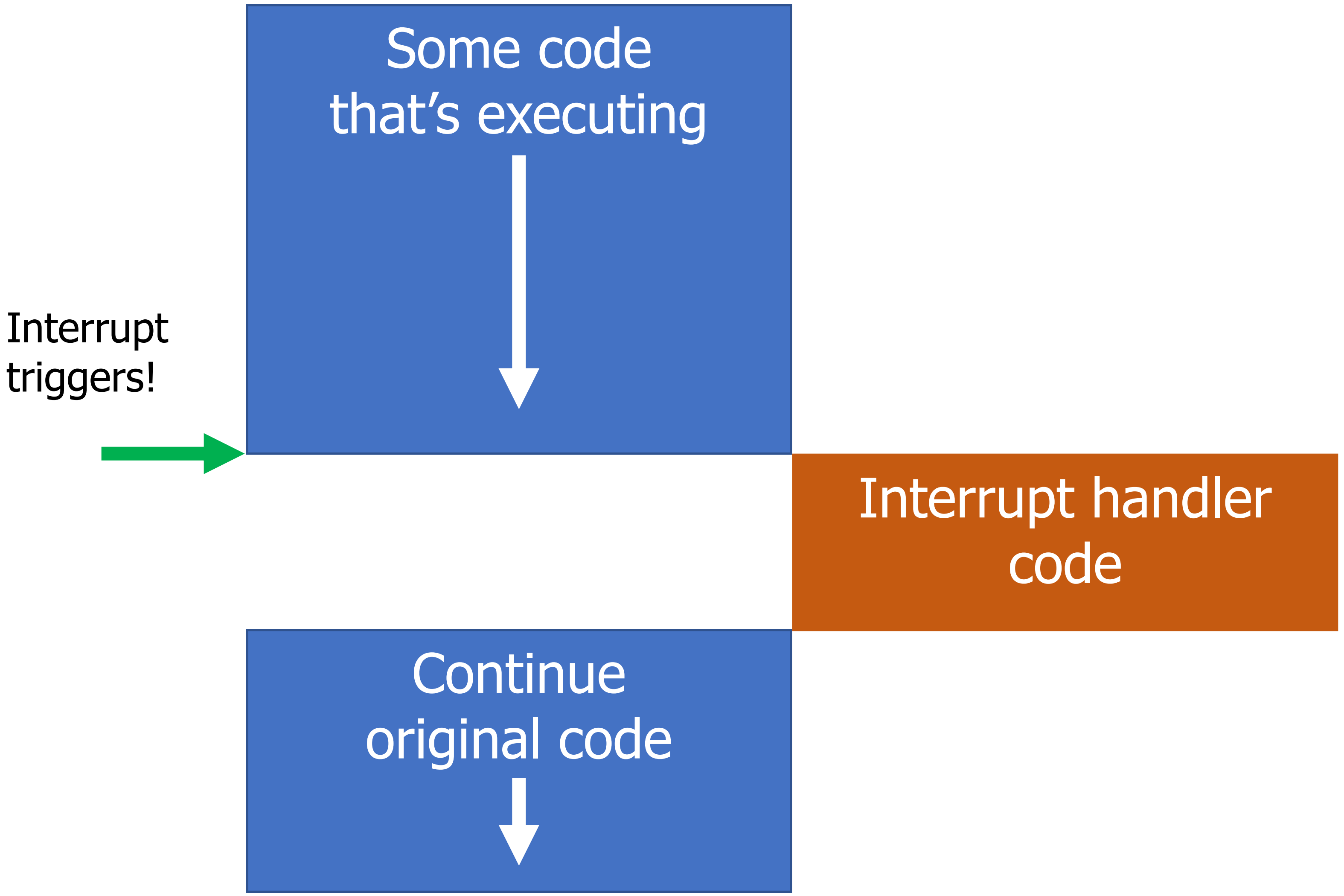
Interrupts, visually



Interrupts, visually



Interrupts, visually



ARM Nested Vectored Interrupt Controller (NVIC)

Interrupts can preempt other interrupts!

Jump directly to the interrupt handler

Handles interrupt entry and exit
- Stacking
- Unstacking
- Priorities

- Manages interrupt requests (IRQ)
  - Stores all caller-saved registers on the stack
    - So the handler code doesn't overwrite them
  - Moves execution to proper handler, a.k.a. Interrupt Service Routine (ISR)
  - Restores registers after handler returns and moves execution back

ARM Vector table

- List of function pointers to handler for each interrupt/exception

- First 15 are architecture-specific exceptions

- After that are microcontroller interrupt signals

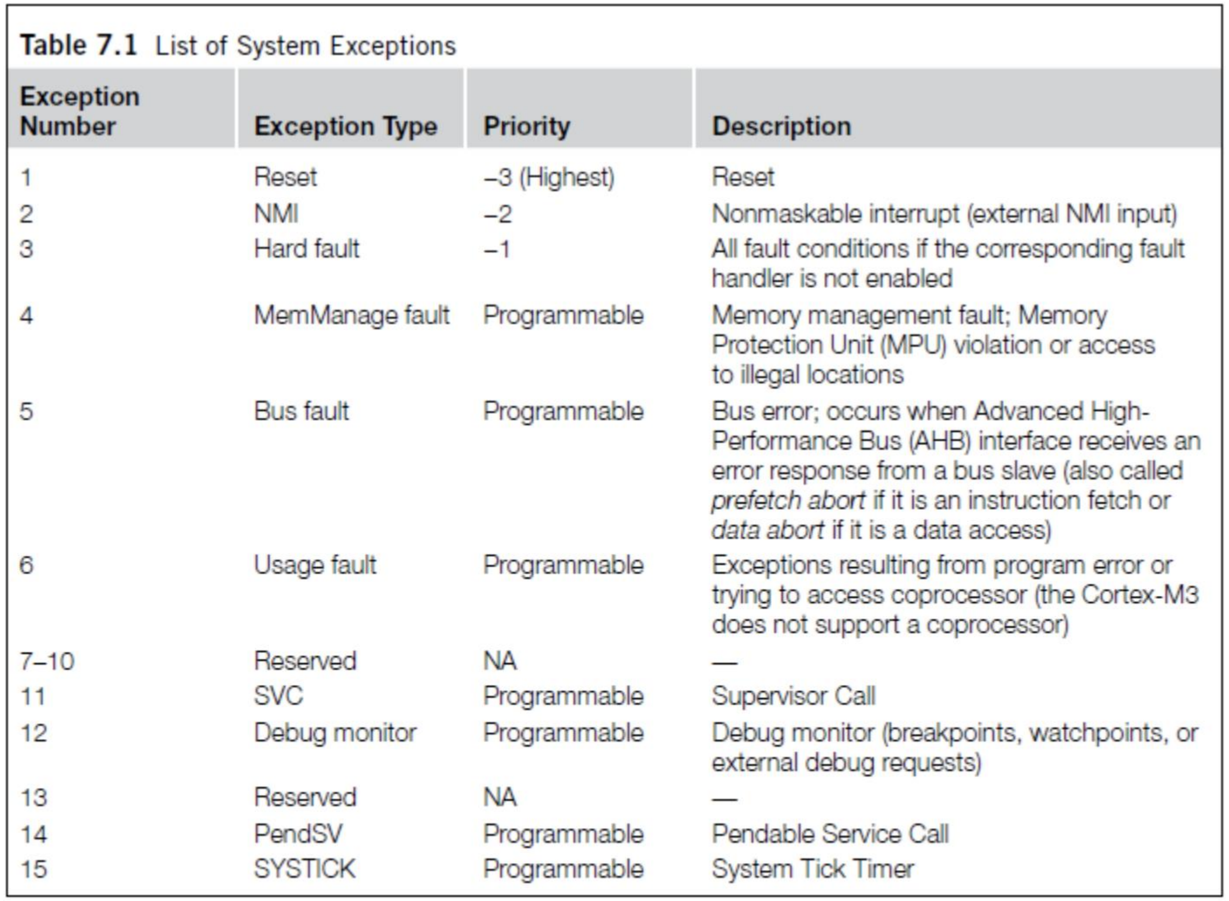


Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt (external NMI input)
3	Hard fault	–1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

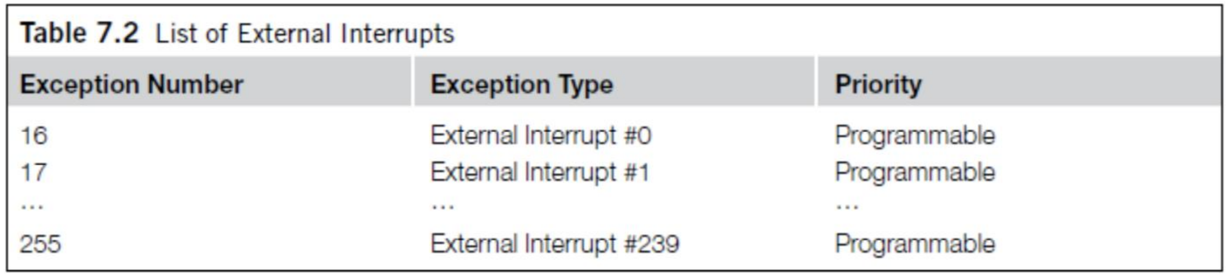
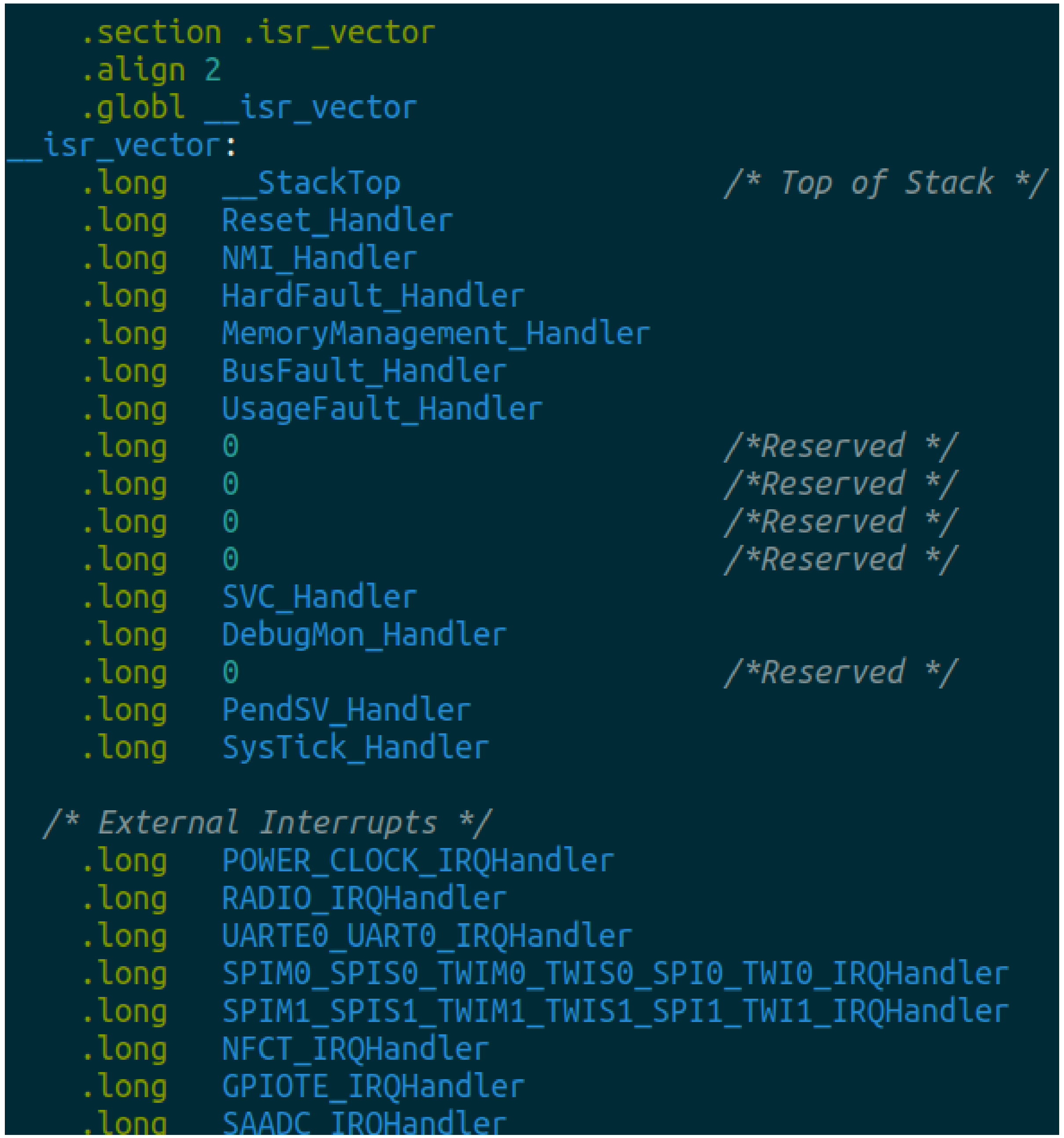


Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
...	...	...
255	External Interrupt #239	Programmable

Vector table in software

- Placed in its own section
  - LD file puts it first in Flash
- Reset_Handler determines where software starts executing
- After that are all exception and interrupt handlers
  - All function pointers to some C code somewhere



```
    .section .isr_vector
    .align 2
    .globl __isr_vector
__isr_vector:
    .long   __StackTop                  /* Top of Stack */
    .long   Reset_Handler
    .long   NMI_Handler
    .long   HardFault_Handler
    .long   MemoryManagement_Handler
    .long   BusFault_Handler
    .long   UsageFault_Handler
    .long   0                           /*Reserved */
    .long   0                           /*Reserved */
    .long   0                           /*Reserved */
    .long   0                           /*Reserved */
    .long   SVC_Handler
    .long   DebugMon_Handler
    .long   0                           /*Reserved */
    .long   PendSV_Handler
    .long   SysTick_Handler

  /* External Interrupts */
    .long   POWER_CLOCK_IRQHandler
    .long   RADIO_IRQHandler
    .long   UARTE0_UART0_IRQHandler
    .long   SPIM0_SPIS0_TWIM0_TWIS0_SPI0_TWI0_IRQHandler
    .long   SPIM1_SPIS1_TWIM1_TWIS1_SPI1_TWI1_IRQHandler
    .long   NFCT_IRQHandler
    .long   GPIOTE_IRQHandler
    .long   SAADC_IRQHandler
```

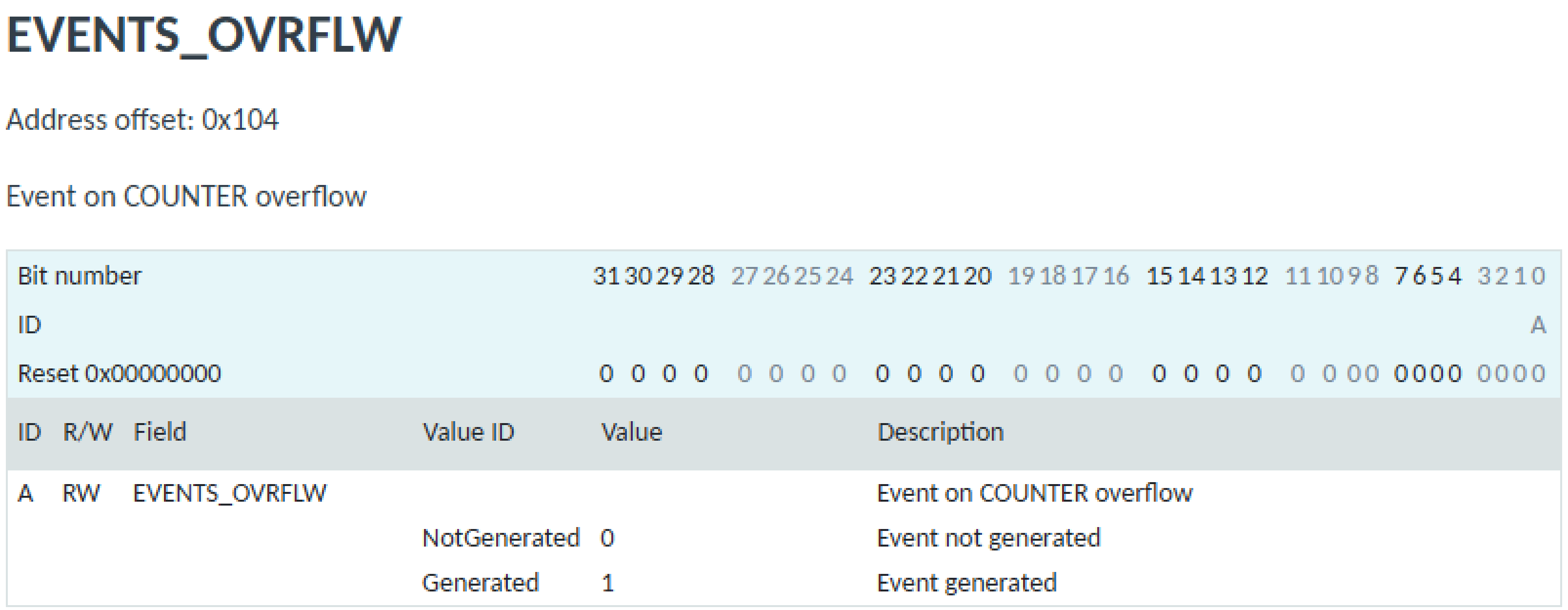
Generating interrupts on the nRF52

- Peripherals can generate “events”
  - 1-bit signal that get set when action occurs
  - There are Event registers that store the value

- Peripherals can enable interrupts
  - Which connects the Event to the NVIC
  - STILL has to be enabled in the NVIC to actually get an Interrupt

- Advanced feature: Events can chain directly into Tasks
  - Tasks are 1-bit actions that a peripheral can take
  - Something occurring can automatically trigger something else to happen
    - PPI is the peripheral that makes these connections
    - (this is advanced stuff though, we’ll rarely use it if ever)

Example "EVENT" register



EVENTS_OVRFLW

Address offset: 0x104

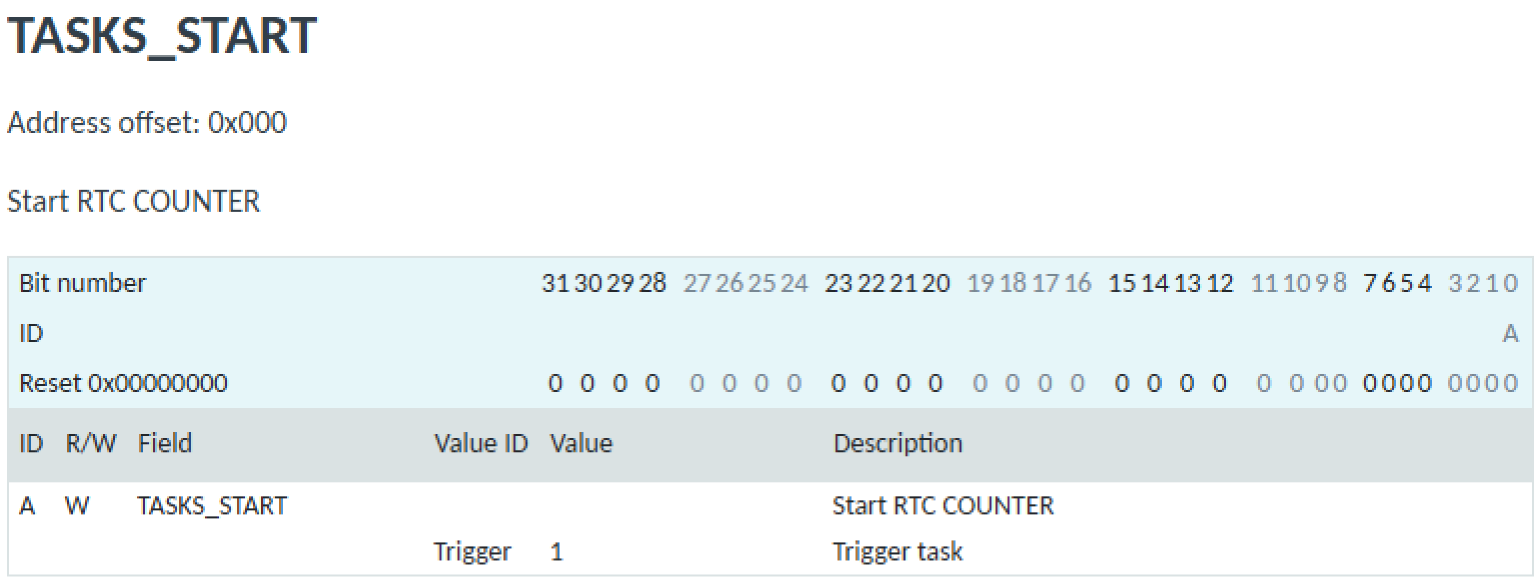
Event on COUNTER overflow

Bit number	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
ID								A
Reset 0x00000000	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

ID	R/W	Field	Value ID	Value	Description
A	RW	EVENTS_OVRFLW			Event on COUNTER overflow
			NotGenerated	0	Event not generated
			Generated	1	Event generated

- nRF-specific concept
  - Event registers record if some action has occurred
  - They are always 1-bit, with a 1 meaning "the action has occurred"
  - Up to the software to clear this register back to 0

Example “TASK” register



TASKS_START

Address offset: 0x000

Start RTC COUNTER

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID																																A
Reset 0x00000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ID	R/W	Field	Value ID	Value	Description
A	W	TASKS_START			Start RTC COUNTER
			Trigger	1	Trigger task

- nRF-specific concept
  - Task registers start some action
  - They are always 1-bit, with a 1 meaning “start the task”

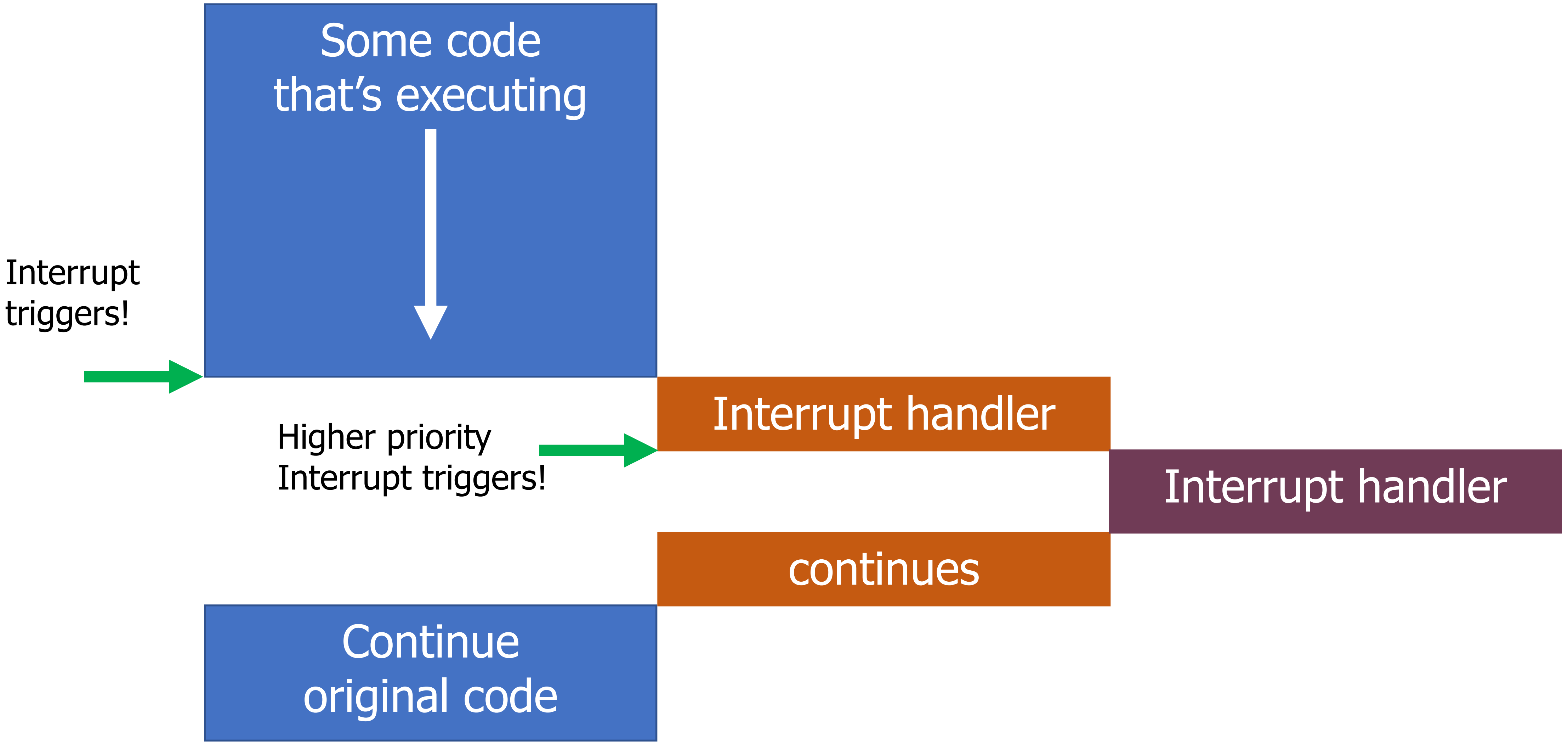
ARM NVIC functionality

- NVIC functions
  - `NVIC_EnableIRQ(number)`
  - `NVIC_DisableIRQ(number)`
  - `NVIC_SetPriority(number, priority)`
    - Technically 256 priorities
    - Only 8 are implemented

- Must enable interrupts in two places!
  - Enabling event in the peripheral will generate the signal
  - Enabling interrupt in the NVIC will cause signal to jump to handler

- Priority determines which interrupt goes first
  - And determines how interrupts are nested

Nested interrupts, visually



Break + Open Question

- When should a system use polling versus interrupts?

Break + Open Question

- When should a system use polling versus interrupts?

- Polling
  - Great if the device is going to respond immediately (like 1 cycle)
  - Important if we need to respond very quick (less than a microsecond)

- Interrupts
  - Great if we'll need to wait a long time for status to change
  - Still responds pretty quickly, but not *immediately*
    - Needs to context switch from running code to interrupt handler

Outline

- I/O Motivation
- Pins
- Controlling digital signals: GIPO peripheral
- Interrupts
- **Digital Input Interrupts: GPIOTE peripheral**
- DMA (Direct Memory Access)

Handling interrupts from GPIO

- Separate peripheral, GPIOTE (GPIO Task/Event)
  - Manages up to 8 individual pins (8 “channels”)
    - Can read input pins and trigger Events
    - Also has Tasks for setting/clearing outputs pins

- Unclear to me why this is a separate peripheral
  - Presumably too complicated/expensive to have 42 of them

Configuring individual input interrupts

- Pick an available GPIOTE channel (0-7)

- Configure it
  - Port and Pin number
  - Task (output), Event (input), or Disabled
  - Polarity for input events
    - Low-to-high
    - High-to-low
    - Toggle (both directions)

- Enable interrupts for channel in GPIOTE (and in NVIC!)

- Clear event in interrupt handler
  - Doesn’t happen automatically!

What if you need more than 8 interrupt sources?

- Can trigger interrupts for a “Port event” as well
  - Any pin in the Port can trigger the interrupt
  - Software checks which pin(s) caused the event to occur

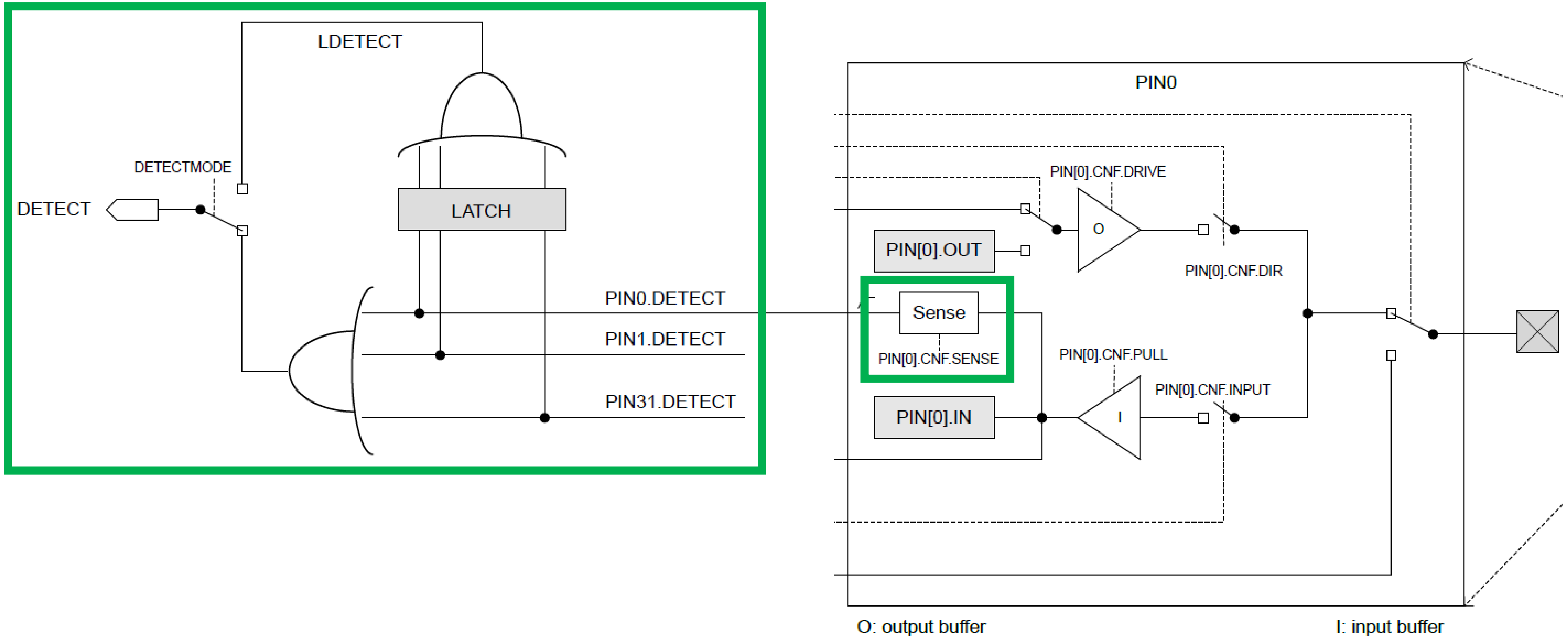
- Very low power operation (works with system clocks off)
  - Whereas normal GPIOTE operations are not low power

Sensing port events

- A Port event is a coordination between GPIO and GPIOTE peripherals
  - GPIO can configure pins with a SENSE option
    - Disabled
    - Trigger on High level
    - Trigger on Low level
  - GPIOTE port event will occur if one or more pins matches their SENSE configuration
    - Software will get an interrupt (if the NVIC is configured)
    - Software checks each pin to determine which one caused the event

Port events can also be “latched”

- Latching saves the value unless cleared



Outline

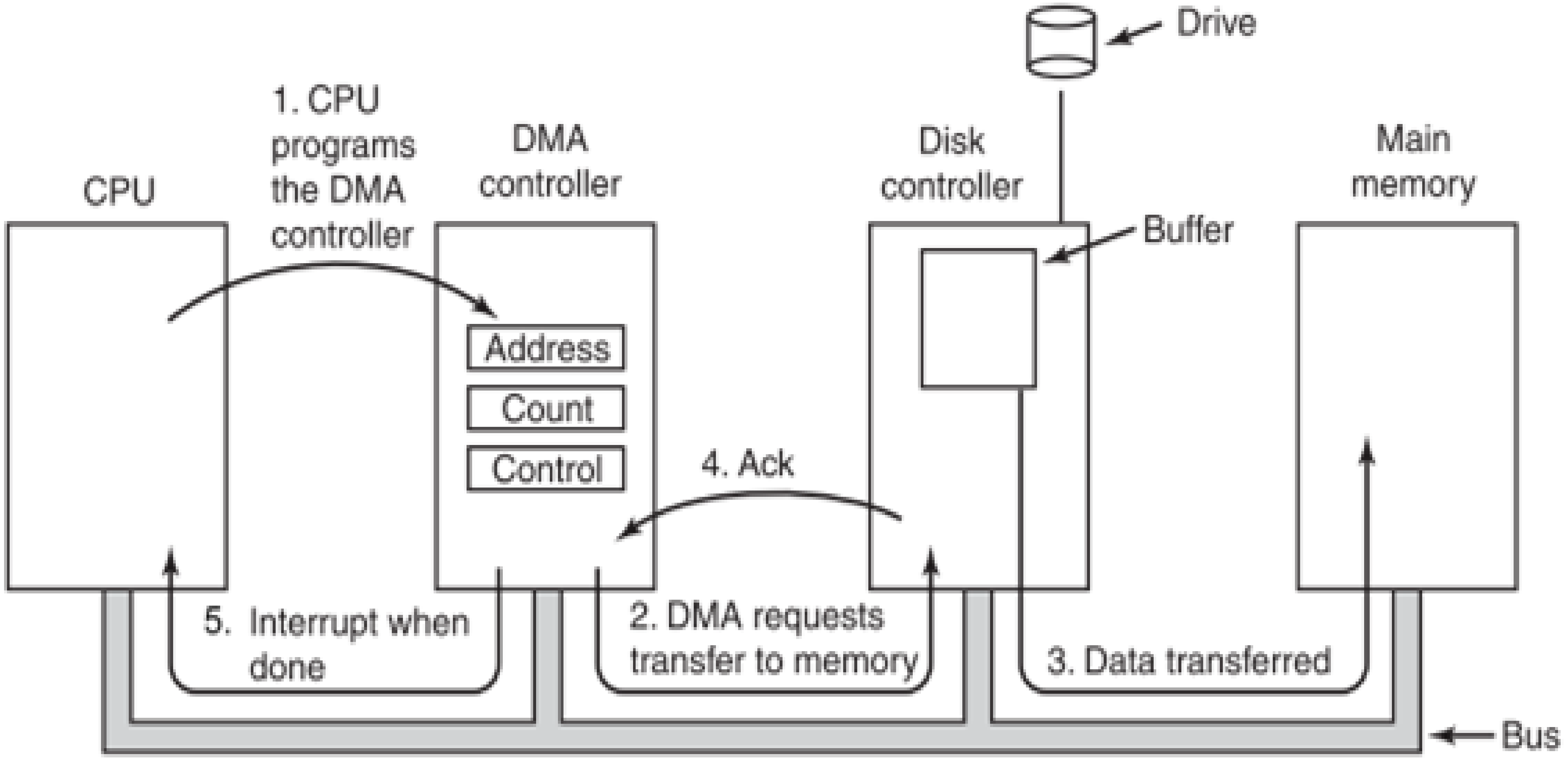
- I/O Motivation
- Pins
- Controlling digital signals: GIPO peripheral
- Interrupts
- Digital Input Interrupts: GPIOTE peripheral
- **DMA (Direct Memory Access)**

Direct Memory Access (DMA)

- Even with interrupts, providing data to the peripheral is time consuming for transferring lots of data
  - Need to be interrupted every byte, to copy the next byte over

- DMA is an alternative method that uses hardware to do the memory transfers for the processor
  - Software writes address of the data and the size to the peripheral
  - Peripheral reads data directly from memory
  - Processor can go do other things while read/write is occurring

General-purpose DMA



Full peripheral interaction pattern

1. Configure the peripheral
2. Enable peripheral interrupts
3. Set up peripheral DMA transfer
4. Start peripheral

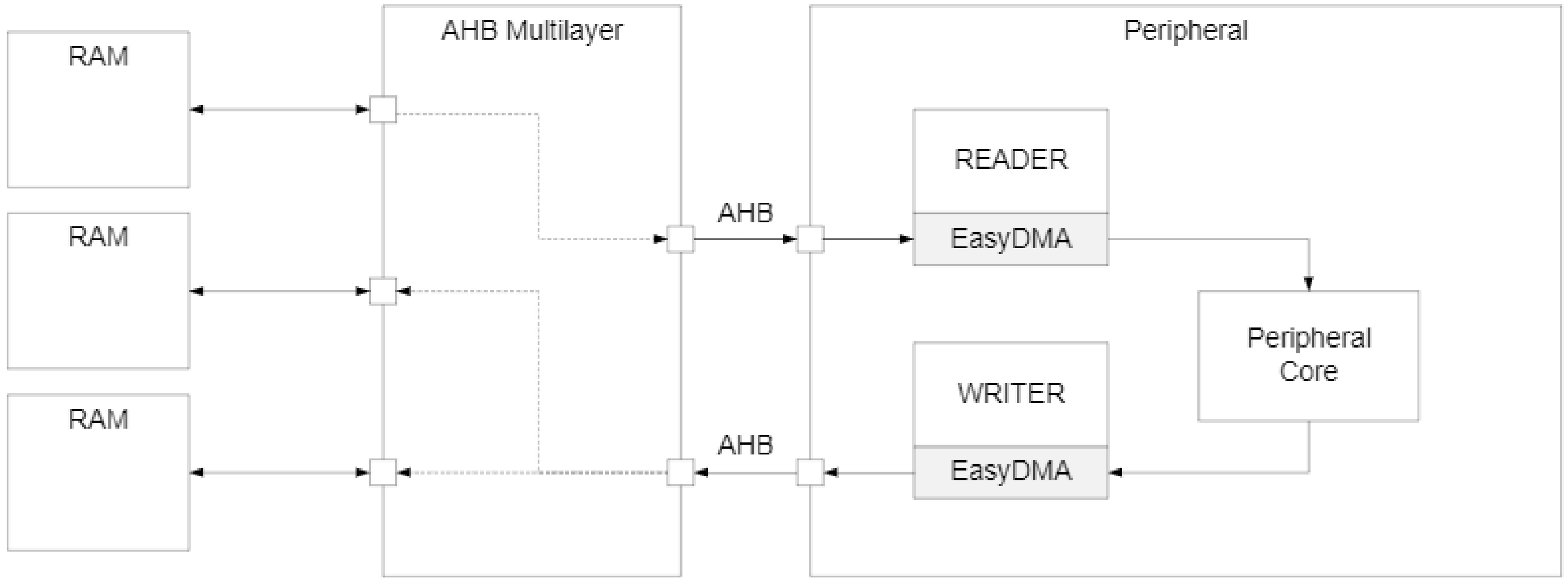
Continue on to other code

5. Interrupt occurs, signaling DMA transfer complete
6. Set up next DMA transfer

Continue on to other code, and repeat

Special-purpose DMA

- nRF52 uses “EasyDMA”, which is built into individual peripherals
  - Only capable of transferring data in/out of that peripheral
  - Easier to set up and use in practice
  - Only available on some peripherals though (no DMA for TEMP or GPIO)



Warning: addresses for DMA buffer MUST be in RAM!

What kinds of peripherals/devices should you use the DMA for?

- Anything where there is a lot of data coming in over a period of time
  - Either a big buffer of lots of data, like a radio message
  - Or a bunch of individual samples, coming in quickly

- Devices
  - Messages to/from other devices (radios, wired busses)
  - Sensor readings (if read quickly/continuously)

  - Canonical example from general computing: disks (HDD/SSD)

Outline

- I/O Motivation
- Pins
- Controlling digital signals: GIPO peripheral
- Interrupts
- Digital Input Interrupts: GPIOTE peripheral
- DMA (Direct Memory Access)