Hardware Acceleration of Cryptography

Patrick Schaumont
Professor

Bradley Department of ECE
Virginia Tech

Outline

1. **Fundamentals of Parallelism**
2. **Embedded Architecture of MSP430, MSP432**
3. **Hardware Acceleration in Embedded Architectures**
4. **AES Hardware Accelerator**
5. **Direct Memory Access**
6. **Power Dissipation**
7. **Literature**

This lecture is about:

- Accelerators in microcontrollers
- Embedded computing
- Efficient crypto (fast, low energy)

This lecture is NOT about:

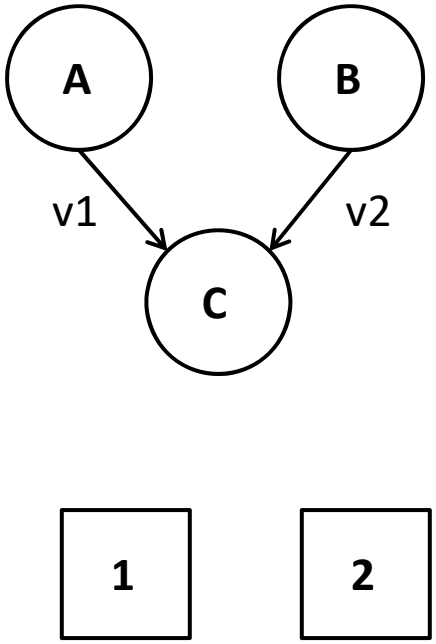
- FPGA
- Multi-core
- High-speed crypto

Outline

1. **Fundamentals of Parallelism**
2. **Embedded Architecture of MSP430, MSP432**
3. **Hardware Acceleration in Embedded Architectures**
4. **AES Hardware Accelerator**
5. **Direct Memory Access**
6. **Power Dissipation**
7. **Literature**

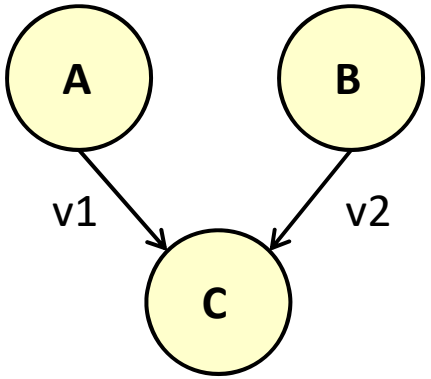
Sequential, Concurrent and Parallel

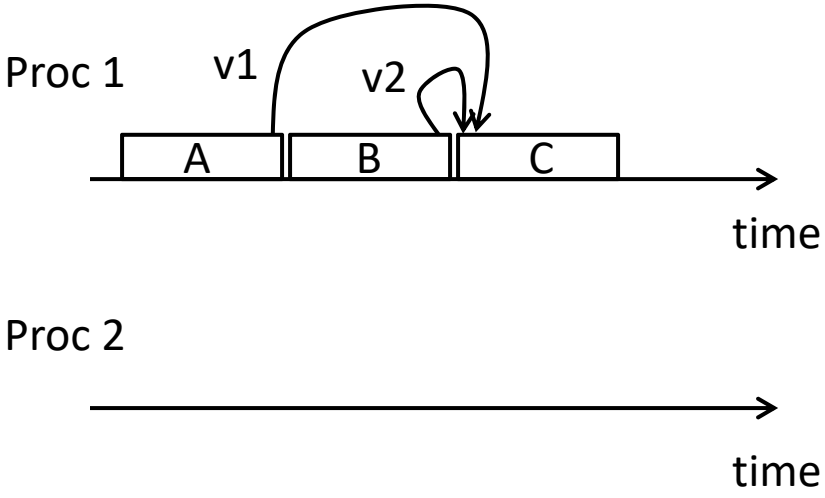
- **Consider three tasks and two processors**

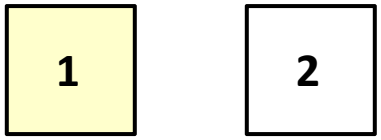


Sequential, Concurrent and Parallel

- **Sequentially running A, B and C on Proc 1**



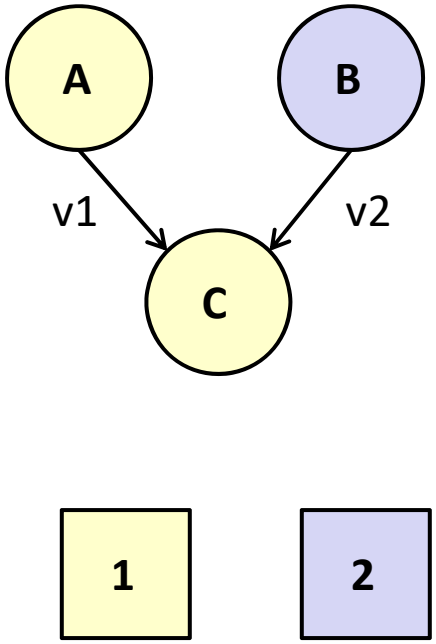


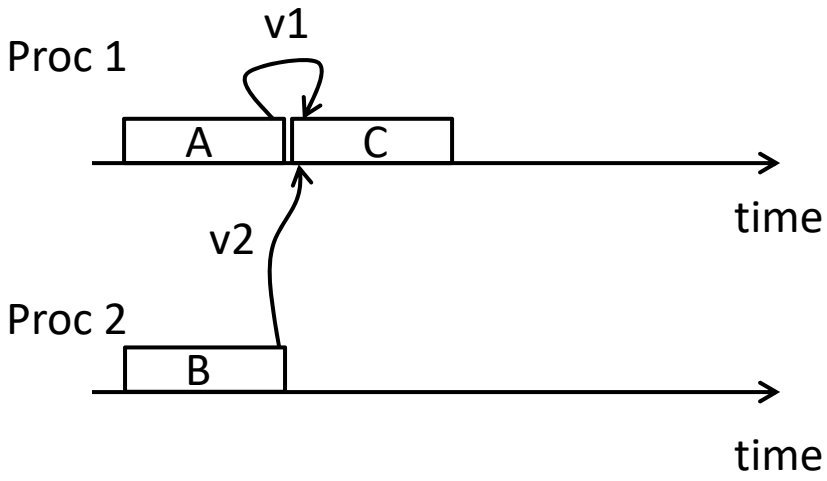


v1 and v2 are stored in Proc 1's memory

Sequential, Concurrent and Parallel

- **In Parallel running A on Proc 1 and B on Proc 2**

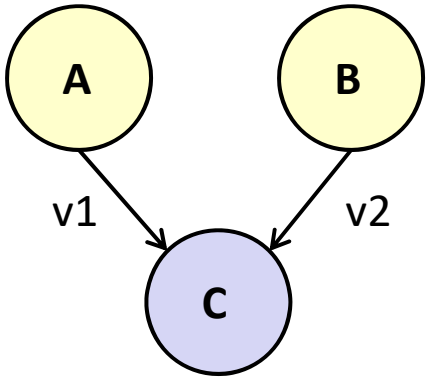


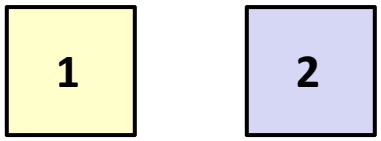


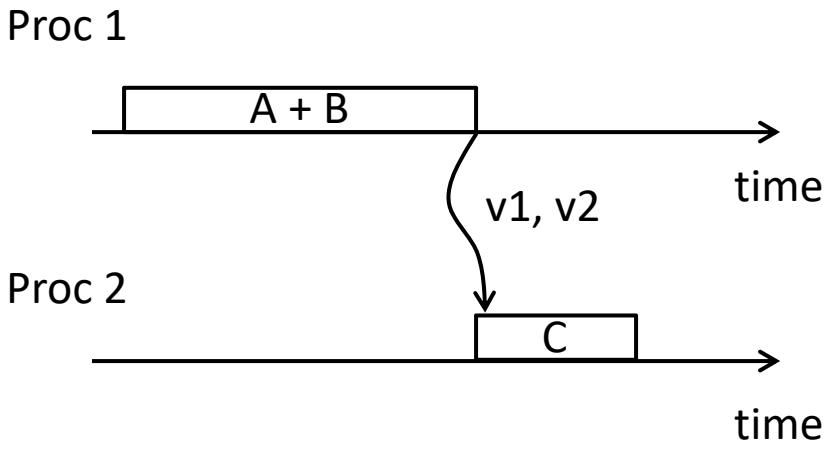
v1 is stored in Proc 1's memory
v2 is communicated from Proc2 to Proc1

Sequential, Concurrent and Parallel

- **Concurrently running A and B on Proc 1, C on Proc 2**





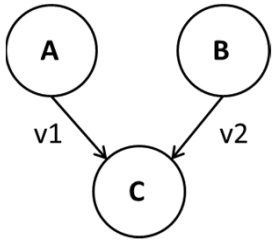


v1 and v2 are communicated from Proc1 to Proc2

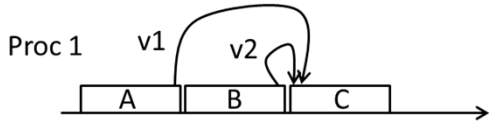
There are many concurrency mechanisms: threading, hyperthreading, SMT, TMT, ..

Control and Data Dependency

A Data Dependency is a relation between two operations such that the result of one operation is used by the next



A Control Dependency is a relation between two operations such that one operation must execute after the other



$A \rightarrow B \rightarrow C$

Control and Data Dependency

A Data Dependency is a relation between two operations such that the result of one operation is used by the next

A Data Dependency is a fundamental property of the application

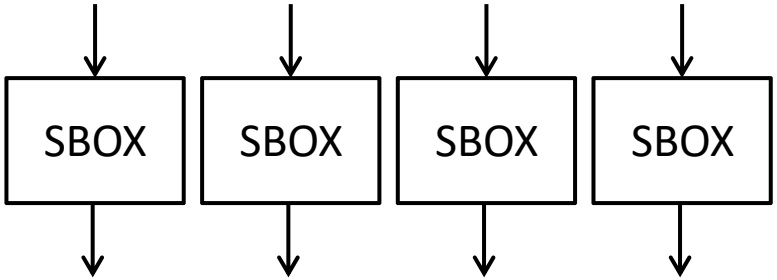
A Control Dependency is a relation between two operations such that one operation must execute after the other

A Control Dependency is caused by a resource constraint

Software and Hardware

Software is sequential/concurrent
Hardware is parallel

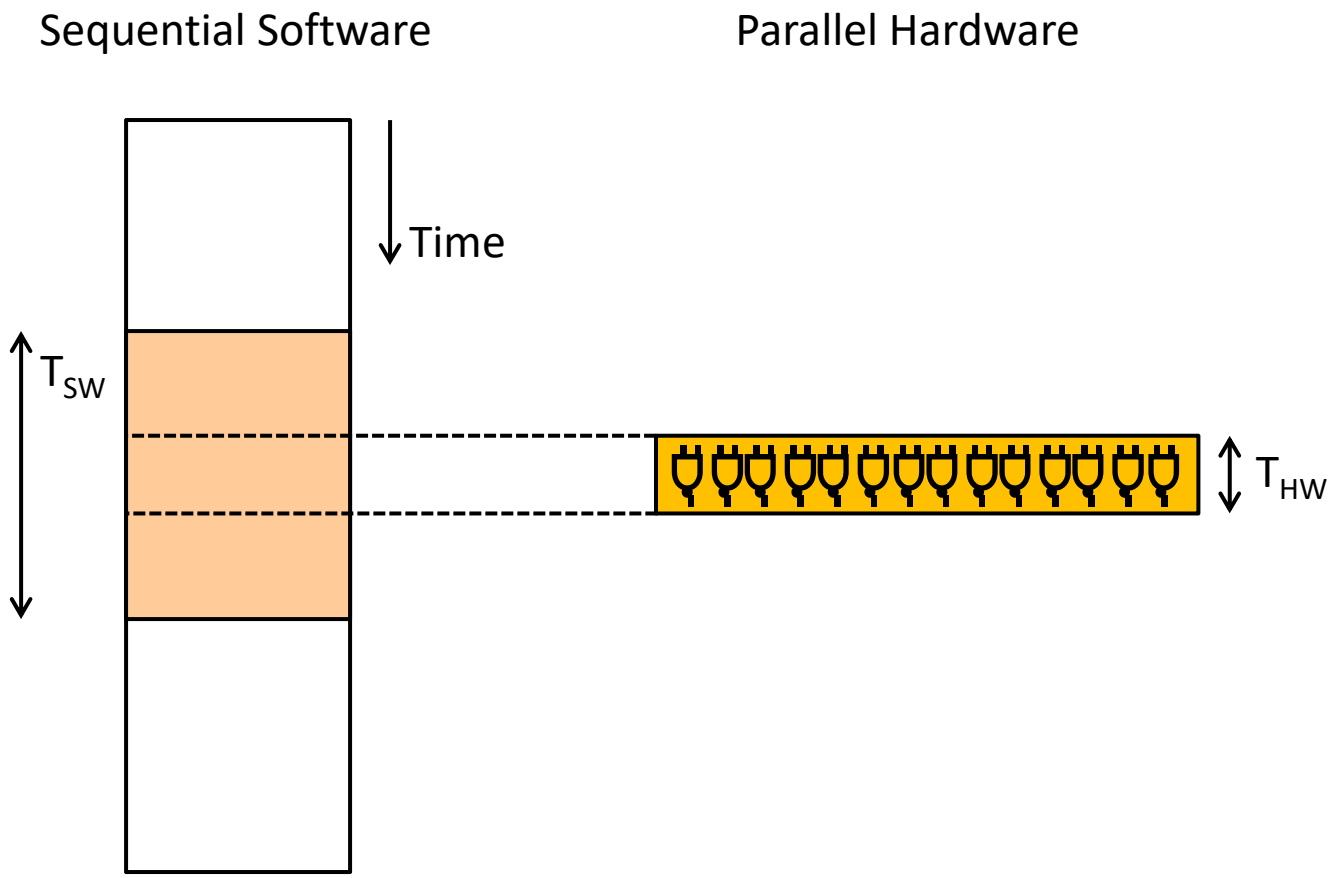
```
PUSH      {A4, V1, V2, LR}
MOVS      A3, #0
MOVW      A4, sbox+0
MOVT      A4, sbox+0
MOVS      A2, #0
ADD       V1, A1, A2, LSL #2
LDRB      V2, [A3, +V1]
LDRB      V2, [A4, +V2]
ADDS      A2, A2, #1
UXTB      A2, A2
CMP       A2, #4
STRB      V2, [A3, +V1]
BLT       ||$C$L7||
ADDS      A3, A3, #1
UXTB      A3, A3
CMP       A3, #4
BLT       ||$C$L6||
POP       {A4, V1, V2, PC}
```



As parallel as allowed by resource constraints
Many control dependencies!

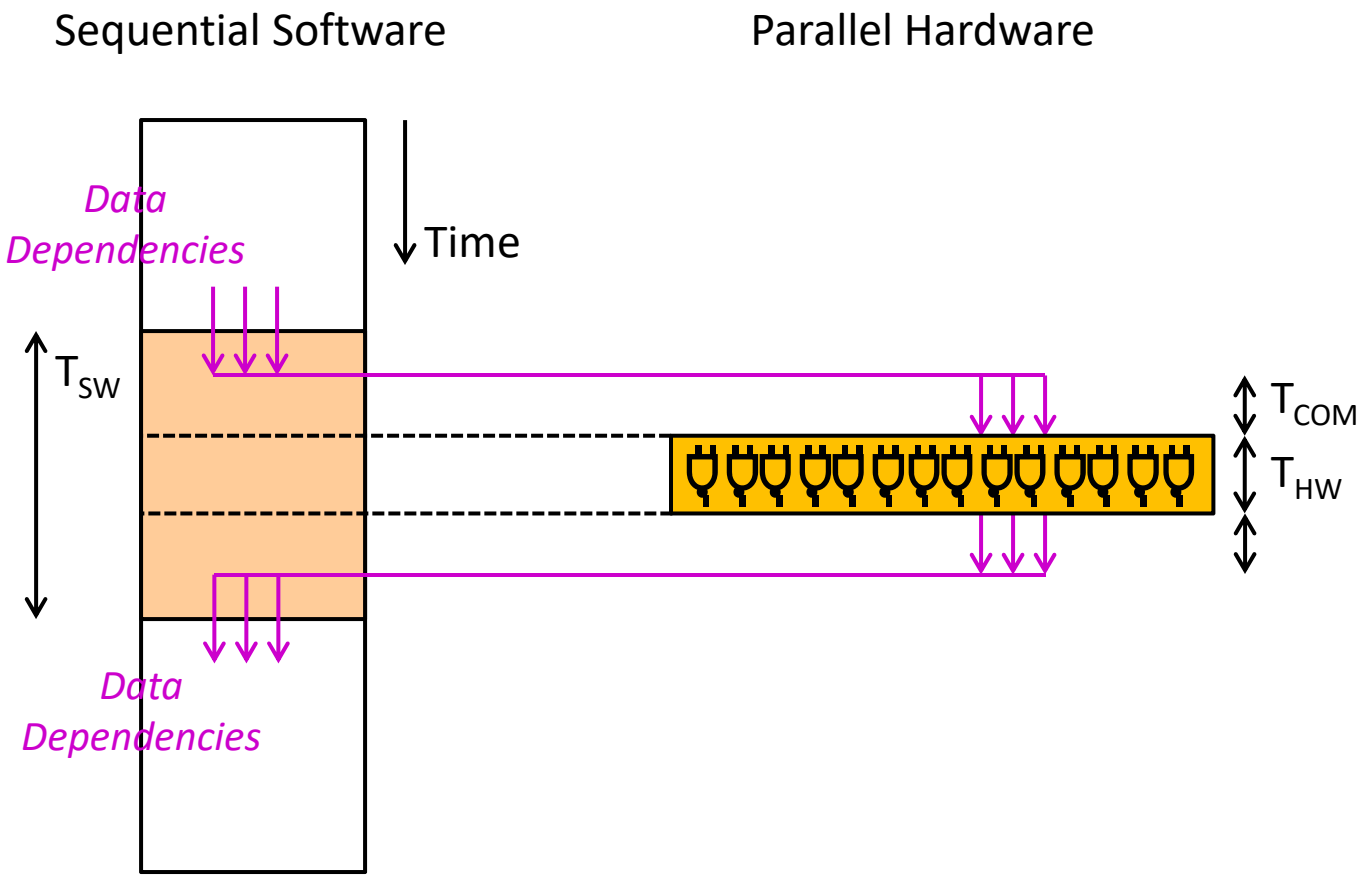
Maximally parallel
Ideally, only data dependencies

Hardware acceleration



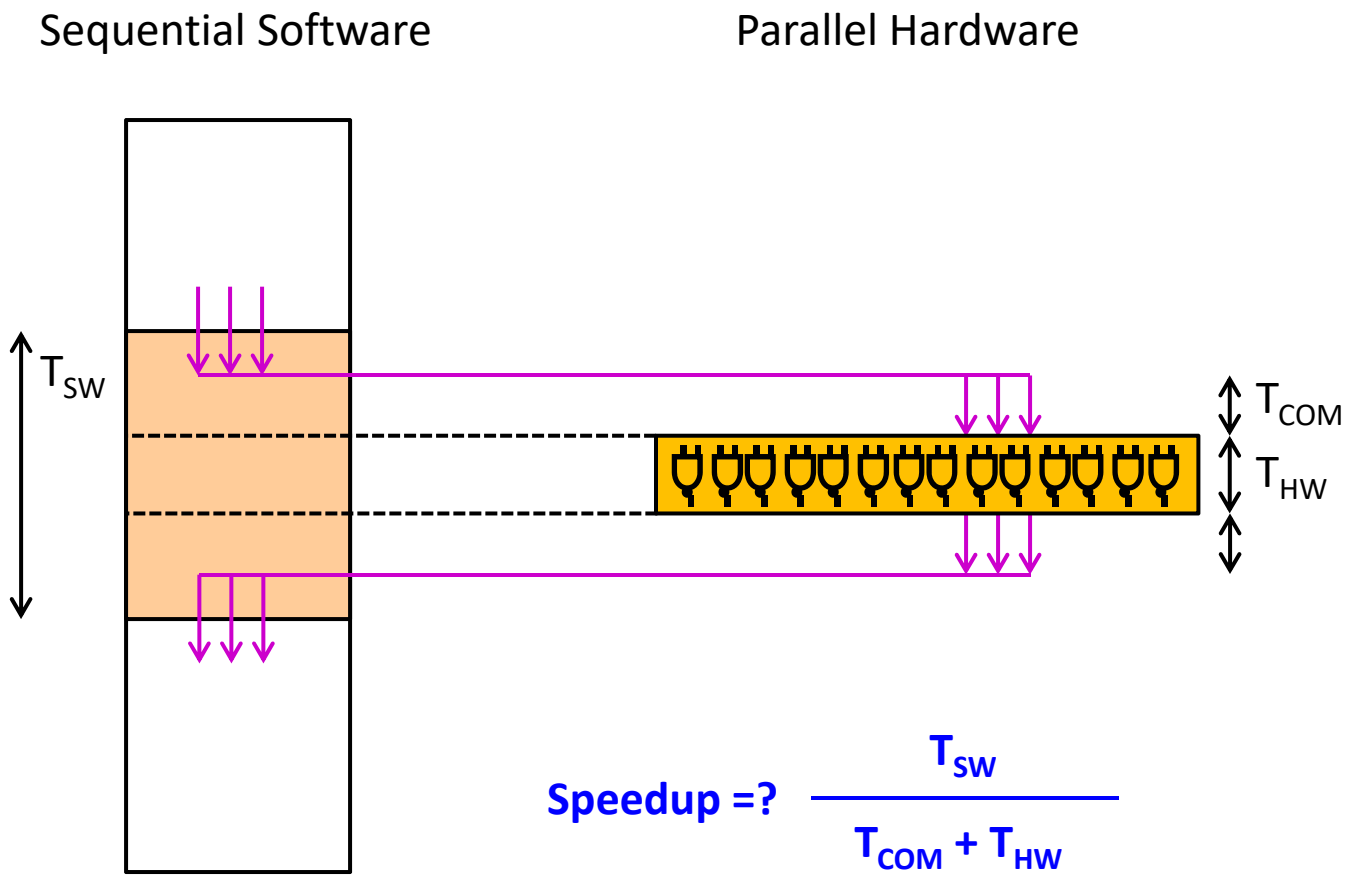
If $T_{HW} < T_{SW}$, overall performance *may* improve

Hardware acceleration

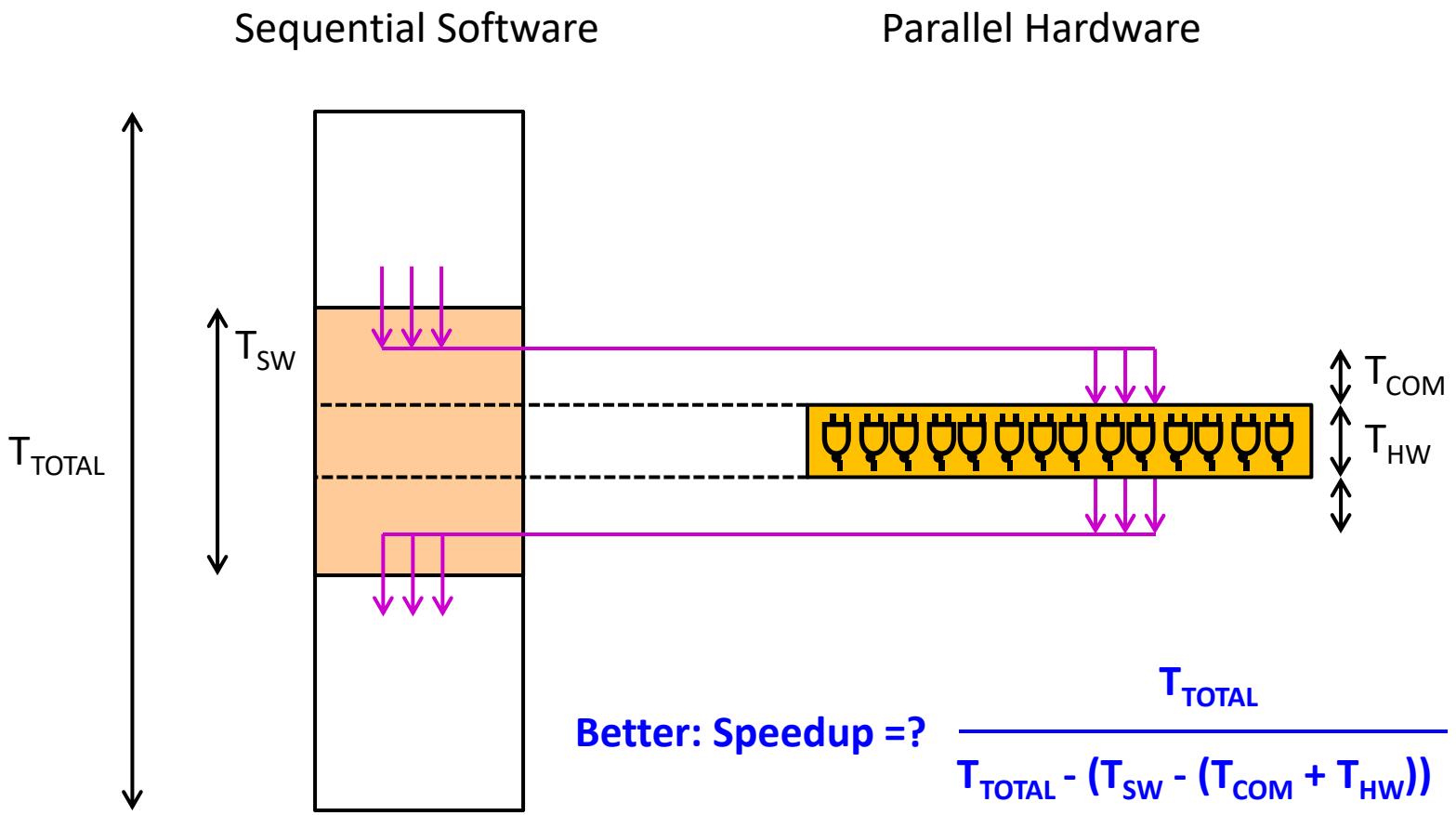


Better: If (T_{HW} + T_{COM}) < T_{SW}, overall performance *may* improve

Speedup


Sequential Software
Parallel Hardware
T_{SW}
T_{COM}
T_{HW}
Speedup =? $\frac{T_{SW}}{T_{COM} + T_{HW}}$

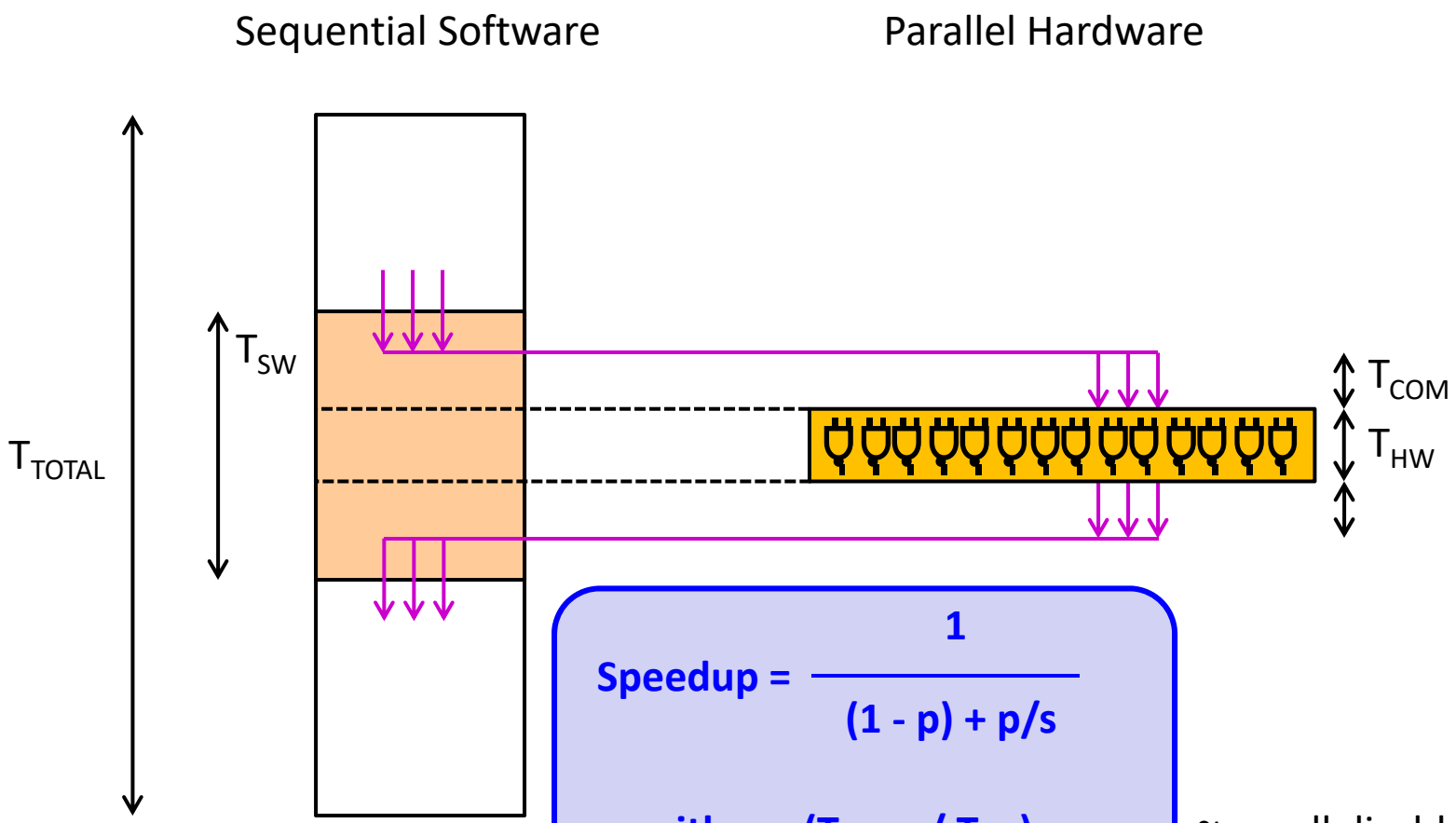
Speedup



Better: Speedup =?

$$\frac{T_{TOTAL}}{T_{TOTAL} - (T_{SW} - (T_{COM} + T_{HW}))}$$

Speedup

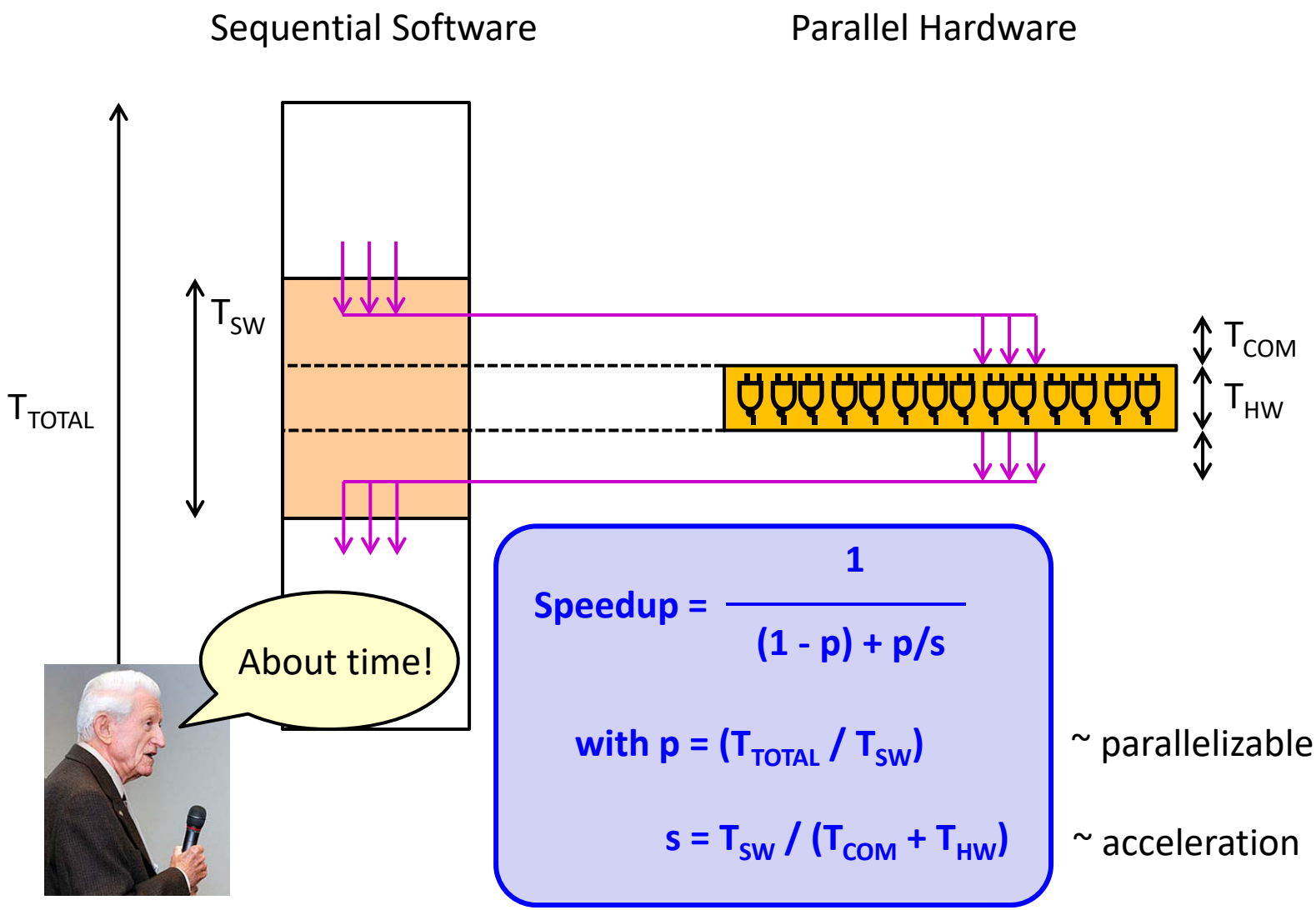


$$\text{Speedup} = \frac{1}{(1 - p) + p/s}$$

with $p = (T_{TOTAL} / T_{SW})$ ~ parallelizable portion

$s = T_{SW} / (T_{COM} + T_{HW})$ ~ acceleration

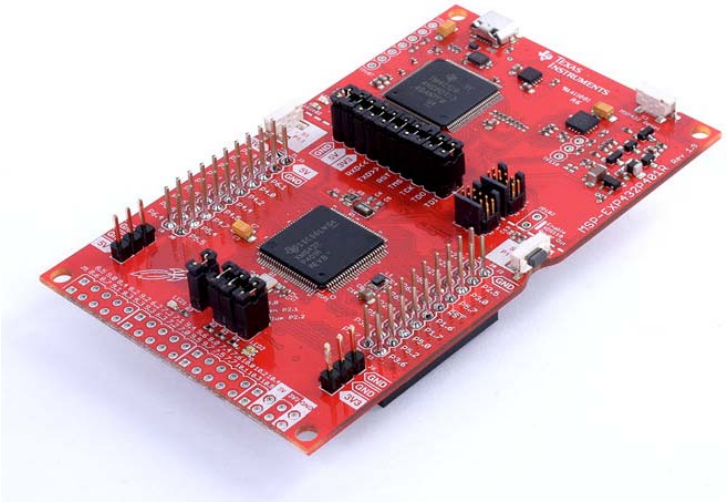
Speedup


Sequential Software
Parallel Hardware
T_{TOTAL}
T_{SW}
T_{COM}
T_{HW}
About time!
Speedup = 1 / ((1 - p) + p/s)
with p = (T_{TOTAL} / T_{SW})
~ parallelizable portion
s = T_{SW} / (T_{COM} + T_{HW})
~ acceleration

Outline

1. Fundamentals of Parallelism
2. Embedded Architecture of MSP430, MSP432
3. Hardware Acceleration in Embedded Architectures
4. AES Hardware Accelerator
5. Direct Memory Access
6. Power Dissipation
7. Literature

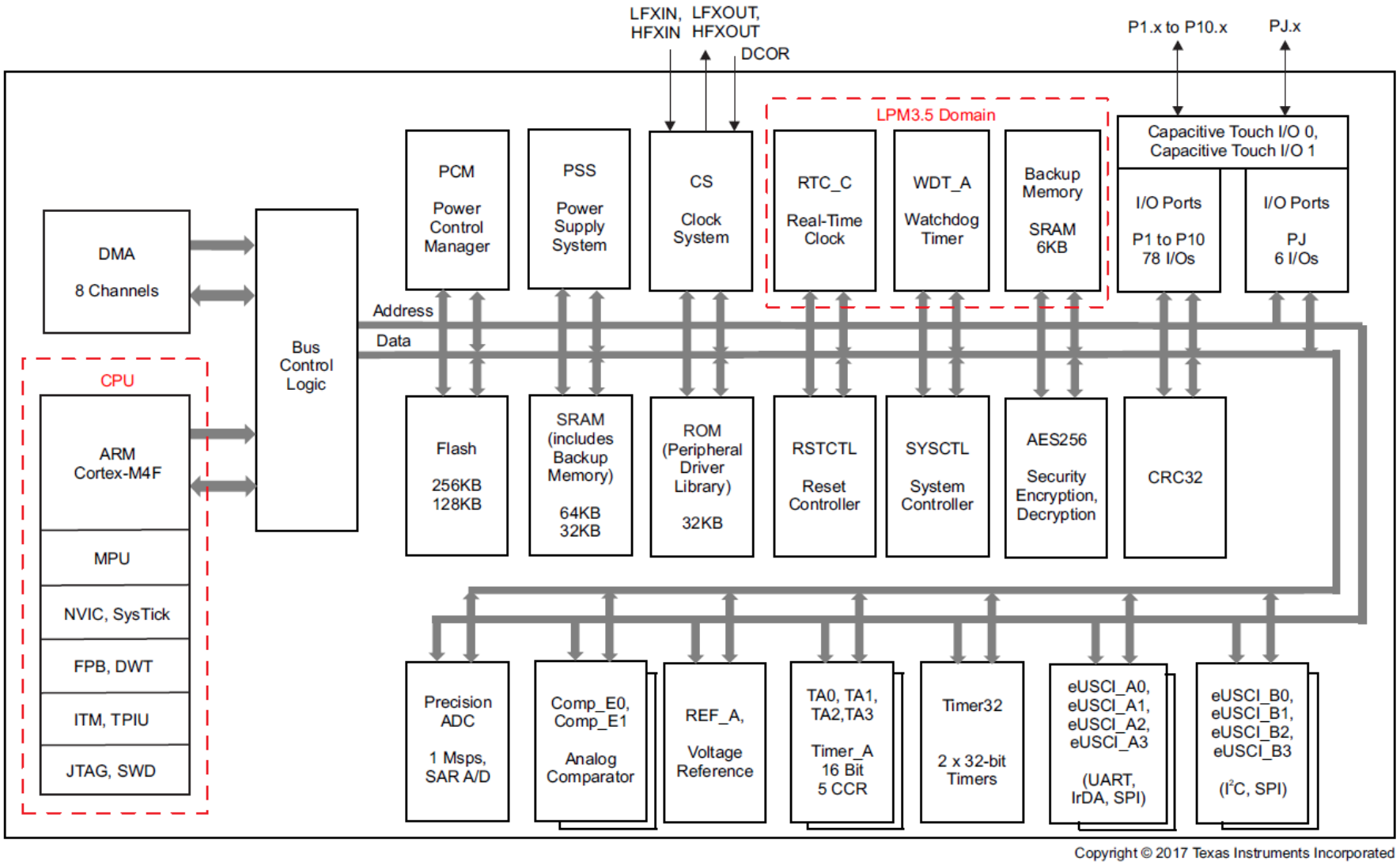
Target Platform



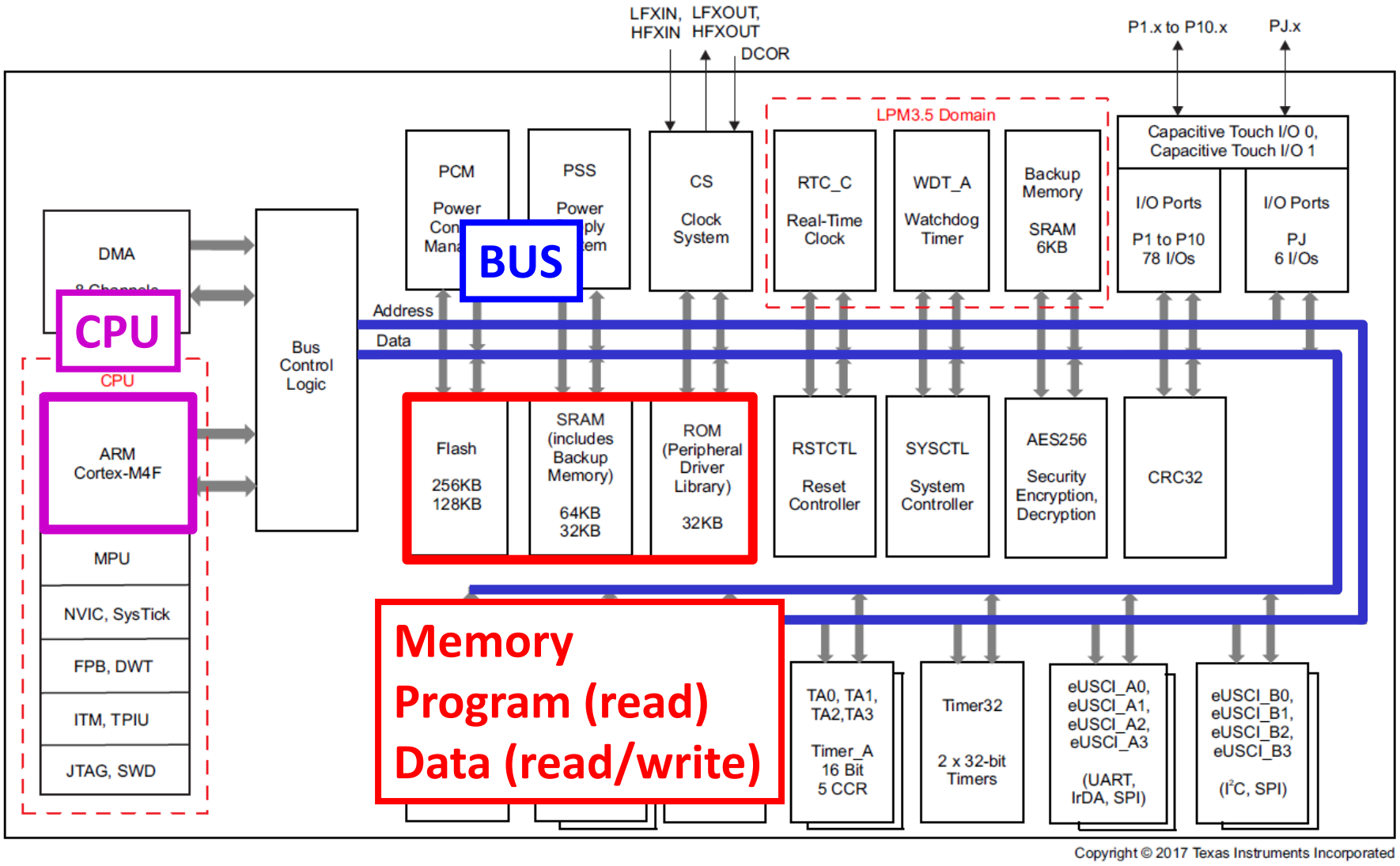
- **MSP432P401R**
- **ARM Cortex M4**
- **256k Flash/64K SRAM**
- **$20**

- **MSP430FR5994**
- **MSP430 (16 bit)**
- **256K FRAM/ 8KSRAM**
- **$17**

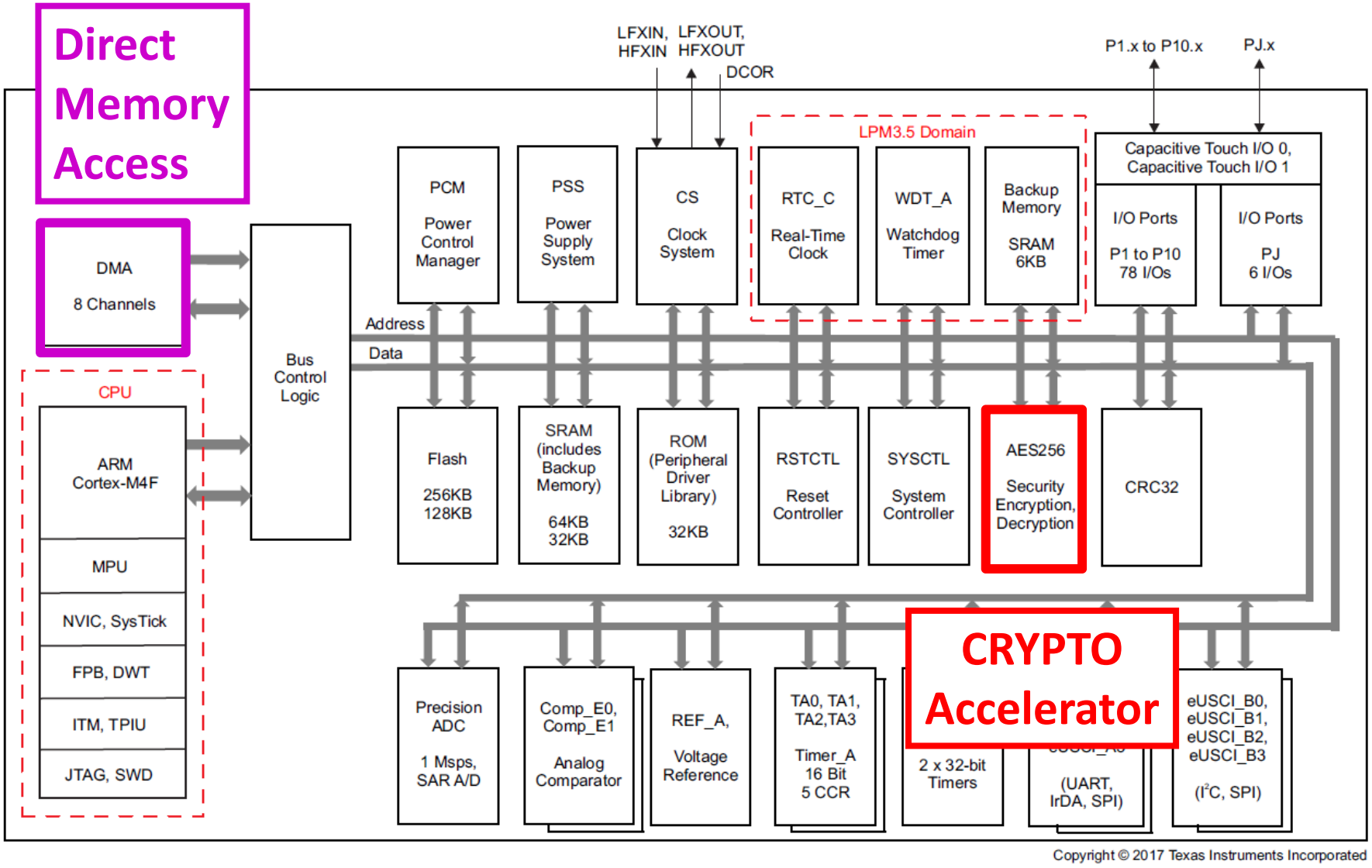
MSP432P401R



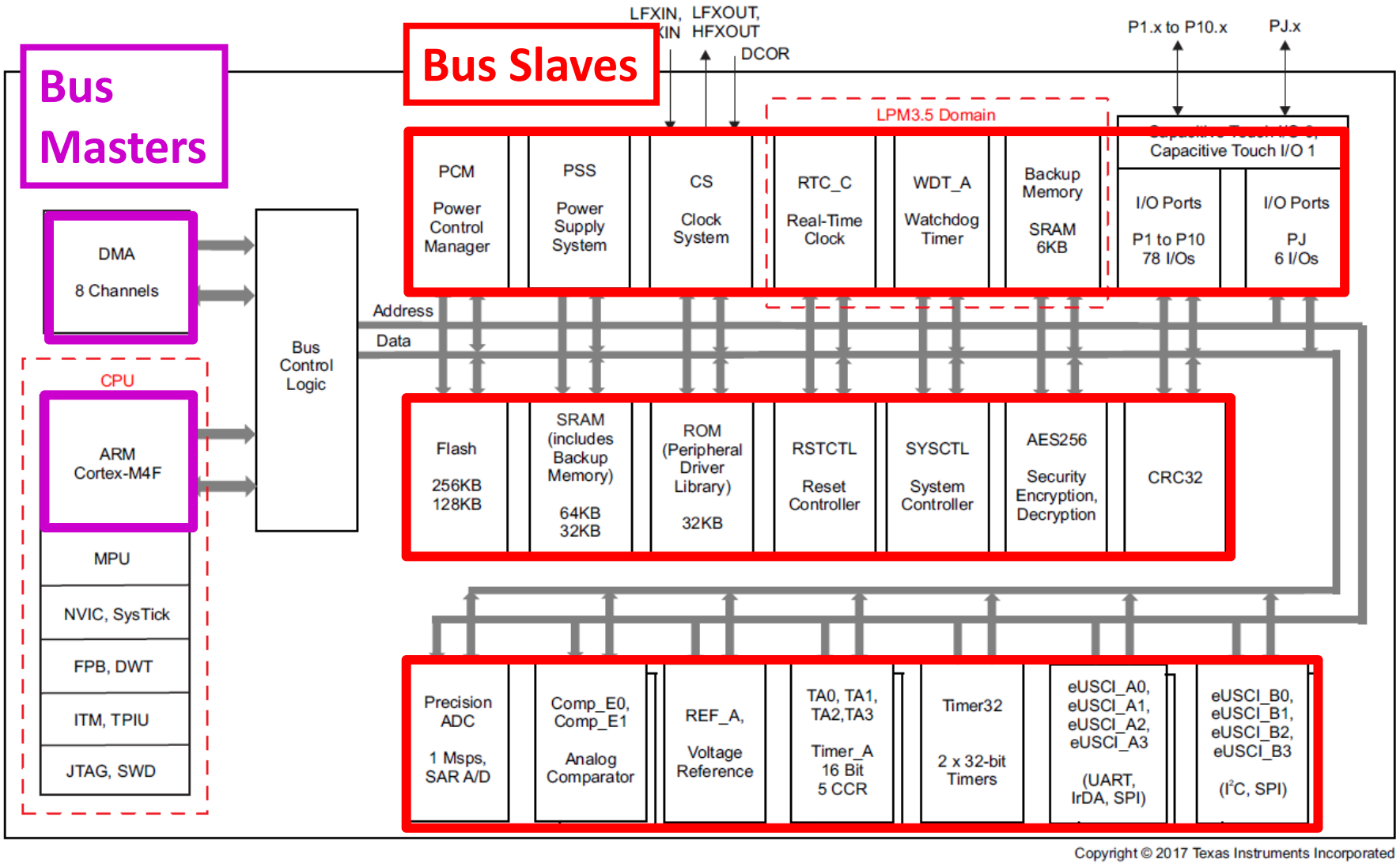
MSP432P401R



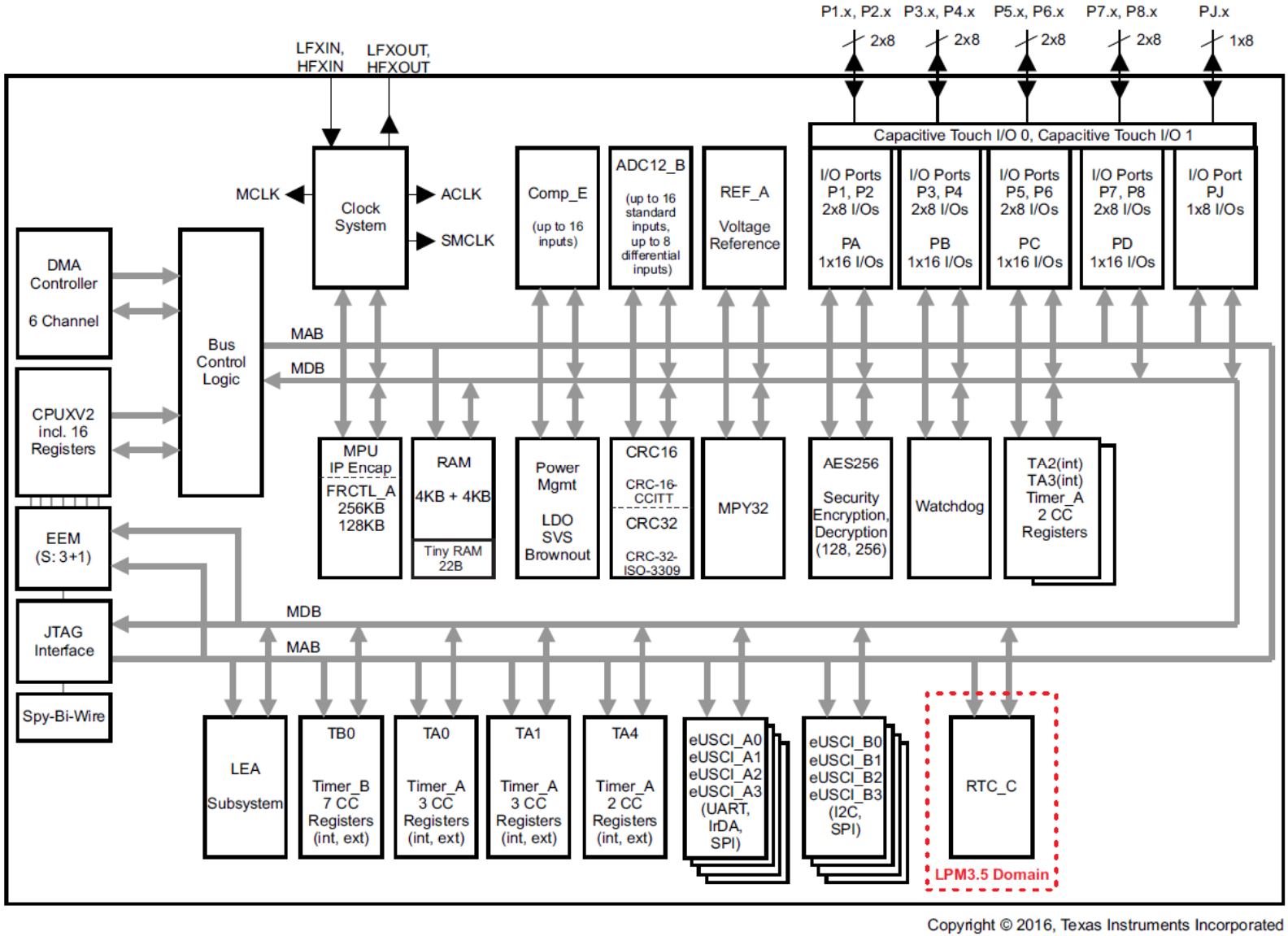
MSP432P401R



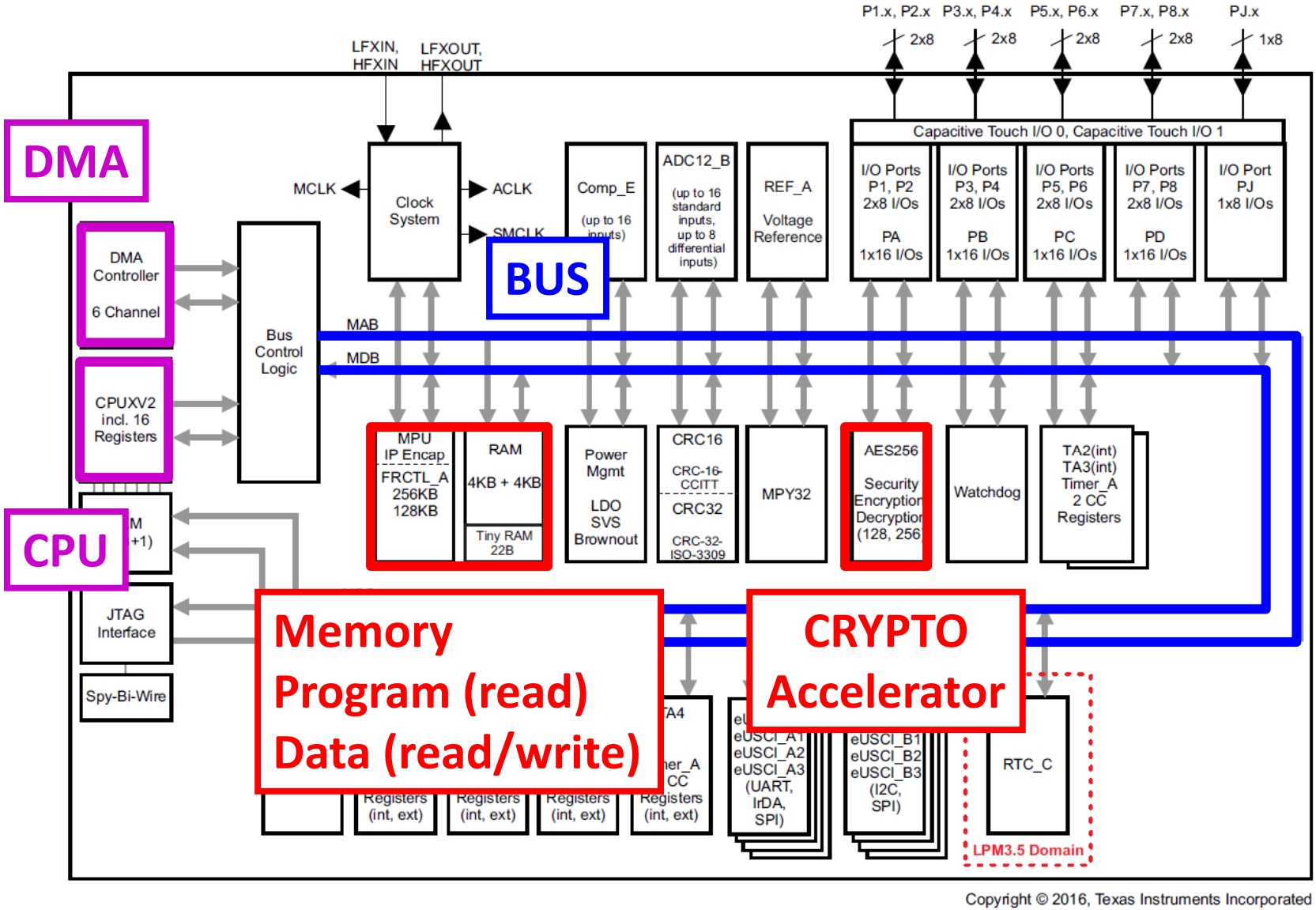
MSP432P401R



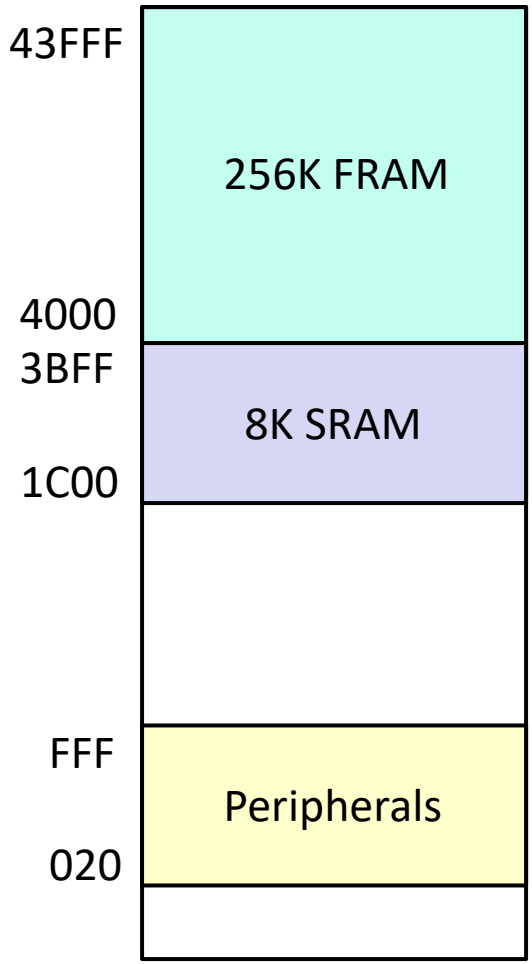
MSP430FR5994



MSP430FR5994

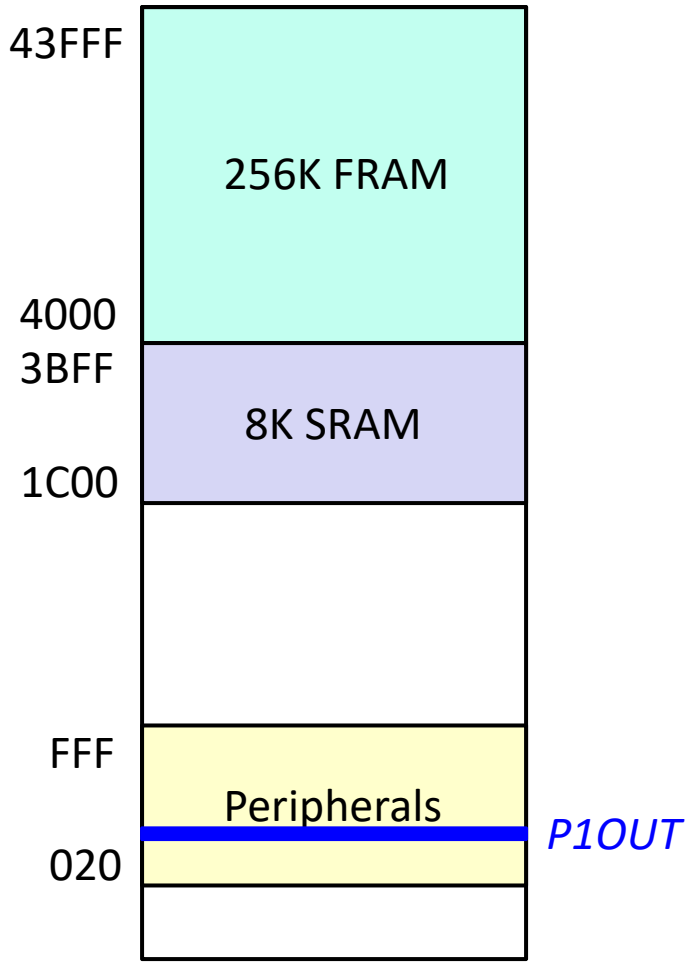


Memory Map (MSP430FR5994)

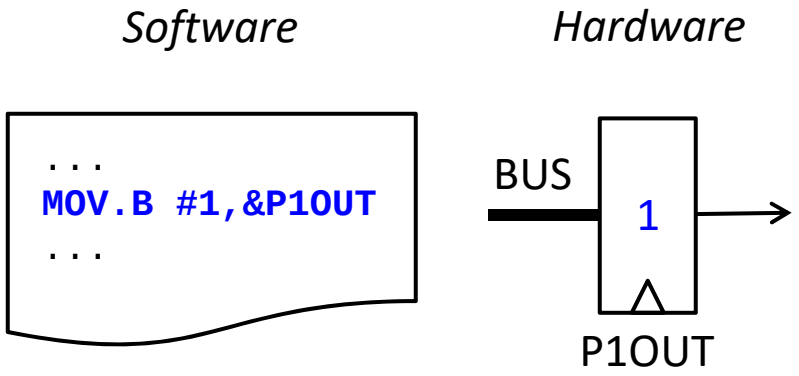


- **Unified view of all bus slaves in a single memory space**
- **FRAM, SRAM store program and variables**
- **Peripherals contain memory-mapped registers**

Memory Map (MSP430FR5994)

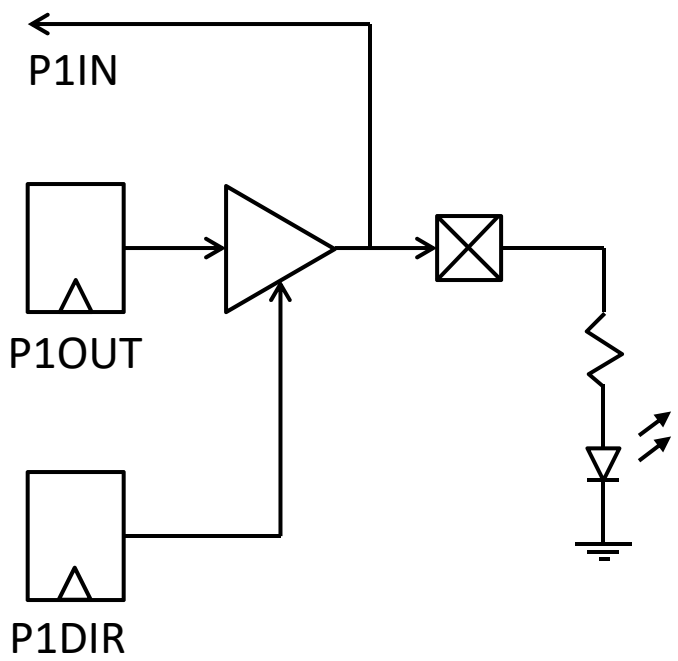


- **Unified view of all bus slaves in a single memory space**
- **FRAM, SRAM store program and variables**
- **Peripherals contain memory-mapped registers**



Example: LED Blinker

```
#include <msp430.h>
int main(void) {
  WDTCTL = WDTPW | WDTHOLD;
  P1OUT &= ~BIT0;
  P1DIR |= BIT0;
  while (1) {
    P1OUT ^= BIT0;
    __delay_cycles(100000);
  }
}

 // Assembly
 BIC.B     #1,&P1OUT+0
 OR.B      #1,&P1DIR+0
 XOR.B     #1,&P1OUT+0
 PUSHM.A   #1, r13
 MOV.W     #33330, r13
 SUB.W     #1, r13
 JNE       $1
 POPM.A    #1, r13
 JMP       $C$L4
```



Outline

1. **Fundamentals of Parallelism**
2. **Embedded Architecture of MSP430, MSP432**
3. **Hardware Acceleration in Embedded Architectures**
4. **AES Hardware Accelerator**
5. **Direct Memory Access**
6. **Power Dissipation**
7. **Literature**

Hardware Acceleration

- **Let's design a multiplier peripheral (MSP430)**
- **Our benchmark program**

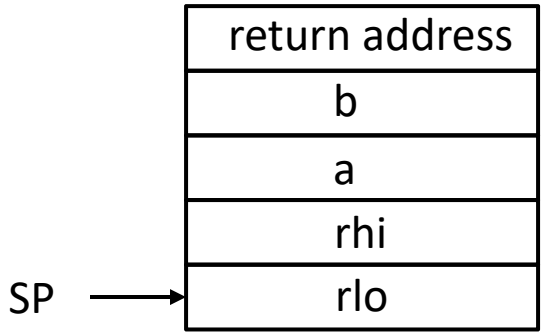
```
unsigned long mymul(unsigned a, unsigned b) {
  unsigned long r;
  r = (unsigned long) a * b;
  return r;
}

volatile int arg1 = 5, arg2 = 3;

int main() {
  return mymul(arg1, arg2);
}
```

Hardware Acceleration

- **Let's design a multiplier peripheral (MSP430)**
- **Our benchmark program**

```
main:
    MOV.W     #5,0(SP)
    MOV.W     #3,2(SP)
    MOV.W     0(SP),r12
    MOV.W     2(SP),r13
    CALLA     #mymul

mymul:
    SUBA      #8,SP
    MOV.W     r13,6(SP)
    MOV.W     r12,4(SP)
    CALLA     #__mspabi_mpyul
    MOV.W     r12,0(SP)
    MOV.W     r13,2(SP)
    ADDA      #8,SP
    RETA
```

arguments



stack frame just before ADDA #8, SP

Hardware Acceleration

- **Our benchmark program**

```
main:
    MOV.W     #5,0(SP)
    MOV.W     #3,2(SP)
    MOV.W     0(SP),r12
    MOV.W     2(SP),r13
    CALLA     #mymul

mymul:
    SUBA      #8,SP
    MOV.W     r13,6(SP)
    MOV.W     r12,4(SP)
    CALLA     #__mspabi_mpyul
    MOV.W     r12,0(SP)
    MOV.W     r13,2(SP)
    ADDA      #8,SP
    RETA
```

arguments

```
__mspabi_mpyul:  // library function
    MOV.W   R12,R11
    MOV.W   R13,R14
    CLR.W   R15
    CLR.W   R12
    CLR.W   R13
    CLRC
    RRC.W   R11
    JMP mpyul_add_loop1
    RRA.W   R11
mpyul_add_loop:
    JNC shift_test_mpyul
mpyul_add_loop1:
    ADD.W   R14,R12
    ADDC.W  R15,R13
    RLA.W   R14
shift_test_mpyul:
    RLC.W   R15
    TST.W   R11
    JNE mpyul_add_loop
    RET
```

Hardware Acceleration

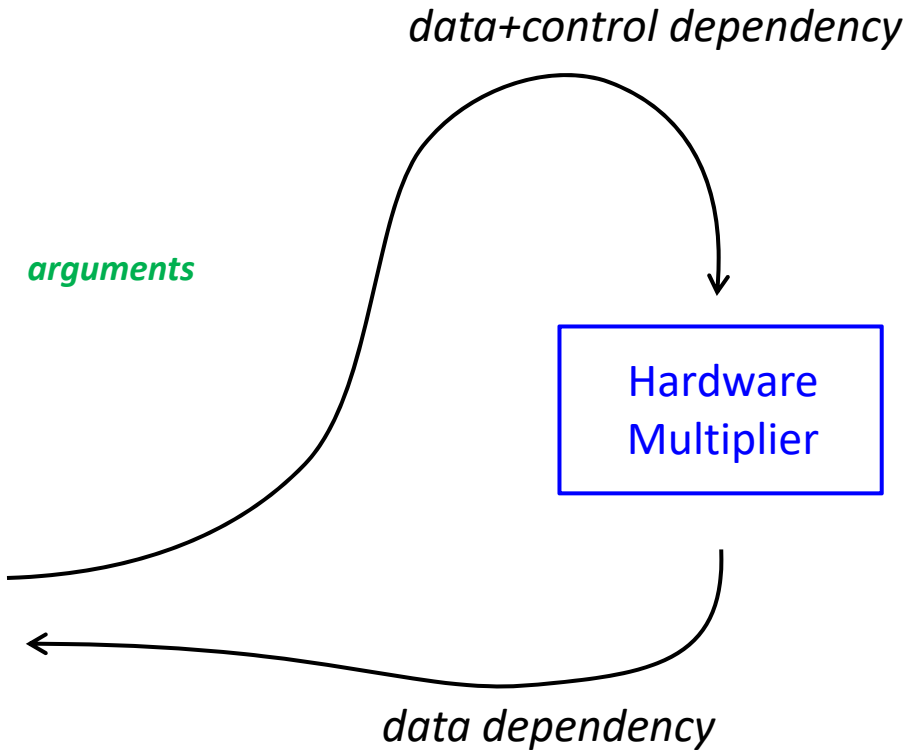
- **Our accelerated program**

```
main:
    MOV.W     #5,0(SP)
    MOV.W     #3,2(SP)
    MOV.W     0(SP),r12
    MOV.W     2(SP),r13
    CALLA     #myhwmul

myhwmul:
```

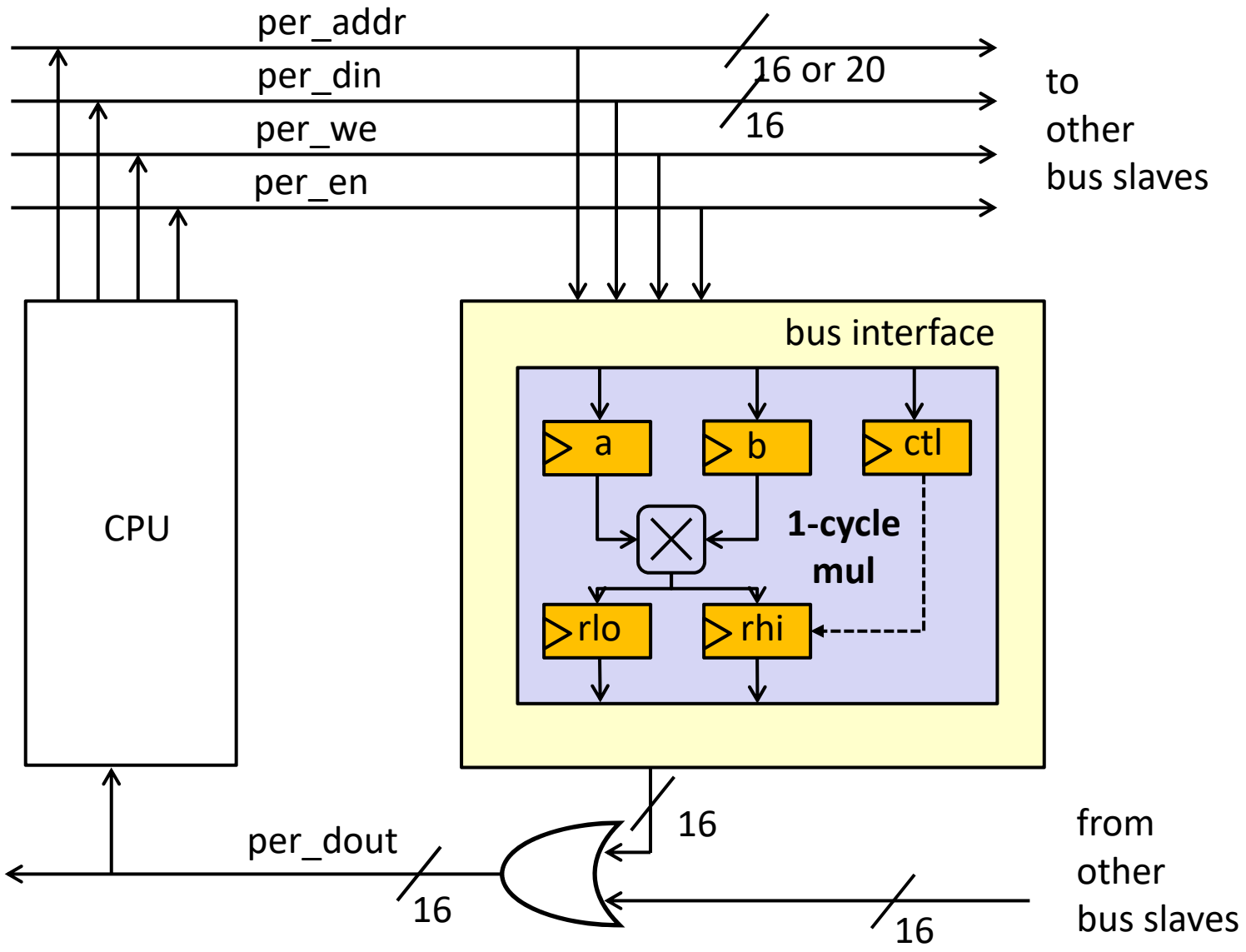
software HAL

```
    MOV.W     r12,&HW1
    MOV.W     r13,&HW2
    MOV.W     #0,&CTL1
    MOV.W     #1,&CTL1
    MOV.W     &HW3,r12
    MOV.W     &HW4,r13
```

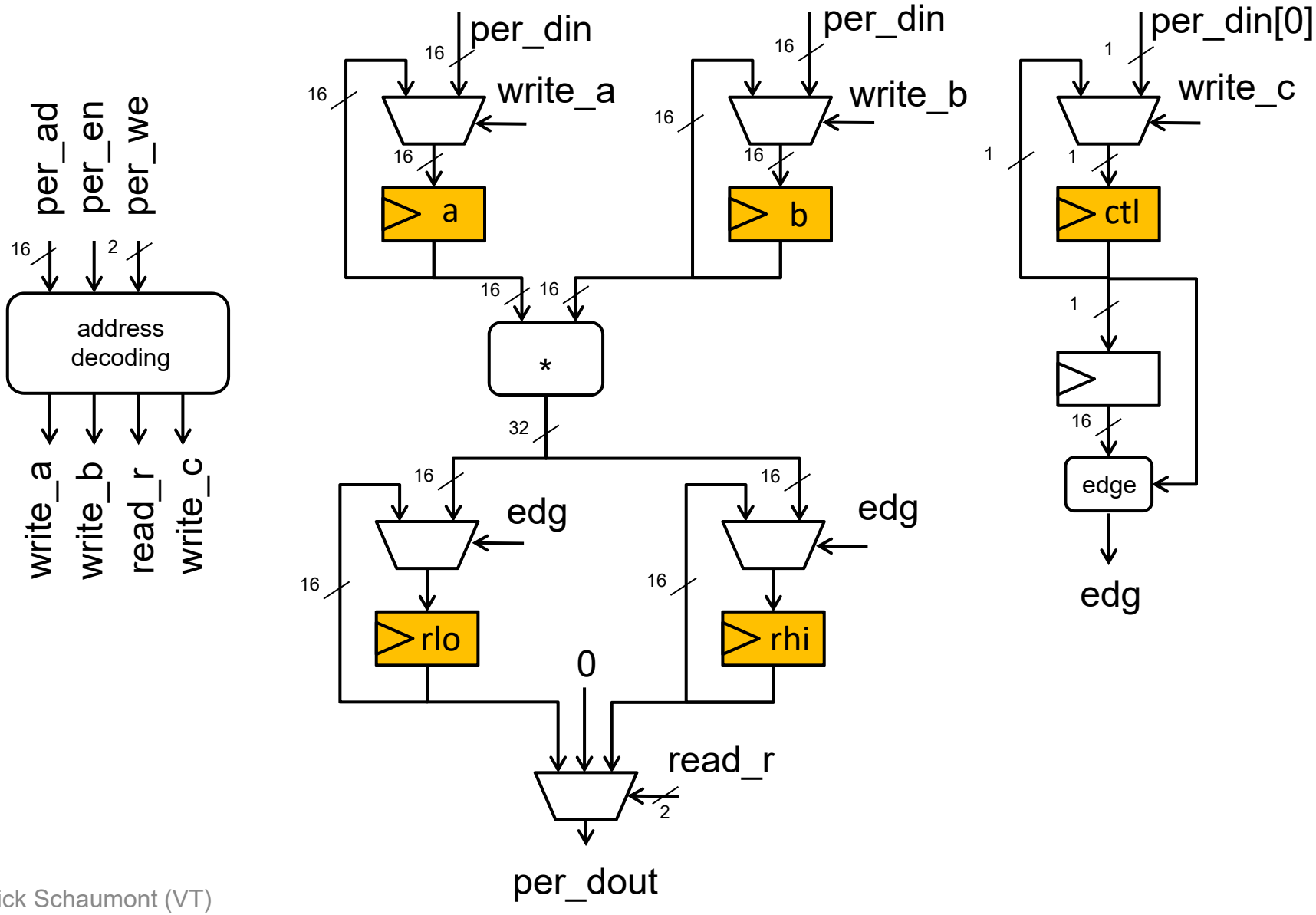
```
    RETA
```



Hardware Multiplier System Interconnect



Hardware Multiplier Peripheral Design



HWMUL Peripheral Design (1)

```
module  mymul ( output [15:0]  per_dout,
                input              mclk,
                input  [13:0]  per_addr,  // word address
                input  [15:0]   per_din,
                input           per_en,
                input  [1:0]     per_we,
                input           puc_rst
                );
reg [15:0]        hw_a, hw_b, hw_retvallo, hw_retvalhi;
reg               hw_ctl, hw_ctl_old;
wire [31:0]       mulresult;
wire              write_a, write_b, write_retval, write_ctl, read_lo, read_hi;

always @(posedge mclk or posedge puc_rst)
begin
  hw_a        <= puc_rst? 16'h0 : write_a      ? per_din[15:0]   : hw_a;
  hw_b        <= puc_rst? 16'h0 : write_b      ? per_din[15:0]   : hw_b;
  hw_retvallo <= puc_rst? 16'h0 : write_retval ? mulresult[15:0] : hw_retvallo;
  hw_retvalhi <= puc_rst? 16'h0 : write_retval ? mulresult[31:0] : hw_retvalhi;
  hw_ctl      <= puc_rst? 16'h0 : write_ctl    ? per_din[0]      : hw_ctl;
  hw_ctl_old  <= hw_ctl;
end

assign mulresult = hw_a * hw_b;
...
```

HWMUL Peripheral Design (2)

```
module  mymul ( output [15:0]  per_dout,
                input              mclk,
                input  [13:0]  per_addr,  // word address
                input  [15:0]   per_din,
                input           per_en,
                input  [1:0]     per_we,
                input           puc_rst
                );
...

assign write_a      = (per_en & (per_addr == 14'hA0) & per_we[0] & per_we[1]);
assign write_b      = (per_en & (per_addr == 14'hA1) & per_we[0] & per_we[1]);
assign write_ctl    = (per_en & (per_addr == 14'hA4) & per_we[0] & per_we[1]);
assign write_retval = ((hw_ctl == 1'h1) & (hw_ctl ^ hw_ctl_old));
assign read_lo      = (per_en & (per_addr == 14'hA2) & ~per_we[0] & ~per_we[1]);
assign read_hi      = (per_en & (per_addr == 14'hA3) & ~per_we[0] & ~per_we[1]);

assign per_dout =  read_lo ? hw_retvallo :
                   read_hi ? hw_retvalhi :
                   16'h0;
endmodule
```

HWMUL Software HAL

byte address

```
#define HW_A      (*(volatile unsigned *)      0x140)
#define HW_B      (*(volatile unsigned *)      0x142)
#define HW_RETVAL (*(volatile unsigned long *) 0x144)
#define HW_CTL    (*(volatile unsigned *)      0x148)

unsigned long mymul_hw(unsigned a, unsigned b) {
  HW_A = a;
  HW_B = b;
  HW_CTL = 1;
  HW_CTL = 0;
  return HW_RETVAL;
}
```

Acceleration ?

```
mymul_hw:
SUBA      #4,SP           // 2
MOV.W     r13,2(SP)       // 4
MOV.W     r12,0(SP)       // 4
MOV.W     0(SP),&0x140    // 6
MOV.W     2(SP),&0x142    // 6
MOV.W     #1,&0x148       // 5
MOV.W     #0,&0x148       // 5
MOV.W     &0x144,r12      // 3
MOV.W     &0x146,r13      // 3
ADDA      #4,SP           // 2
RETA                      // 2
```

Hardware Multiply
1 cycle

HAL Multiply
42 cycles

Acceleration ?

Ideal Speedup (SW-HW) **122x**

Practical Speedup (SW-HAL) **2.9x**

```
mymul_hw:
SUBA      #4,SP           // 2
MOV.W     r13,2(SP)       // 4
MOV.W     r12,0(SP)       // 4
MOV.W     0(SP),&0x140    // 6
MOV.W     2(SP),&0x142    // 6
MOV.W     #1,&0x148       // 5
MOV.W     #0,&0x148       // 5
MOV.W     &0x144,r12      // 3
MOV.W     &0x146,r13      // 3
ADDA      #4,SP           // 2
RETA                      // 2
```

Hardware Multiply
1 cycle

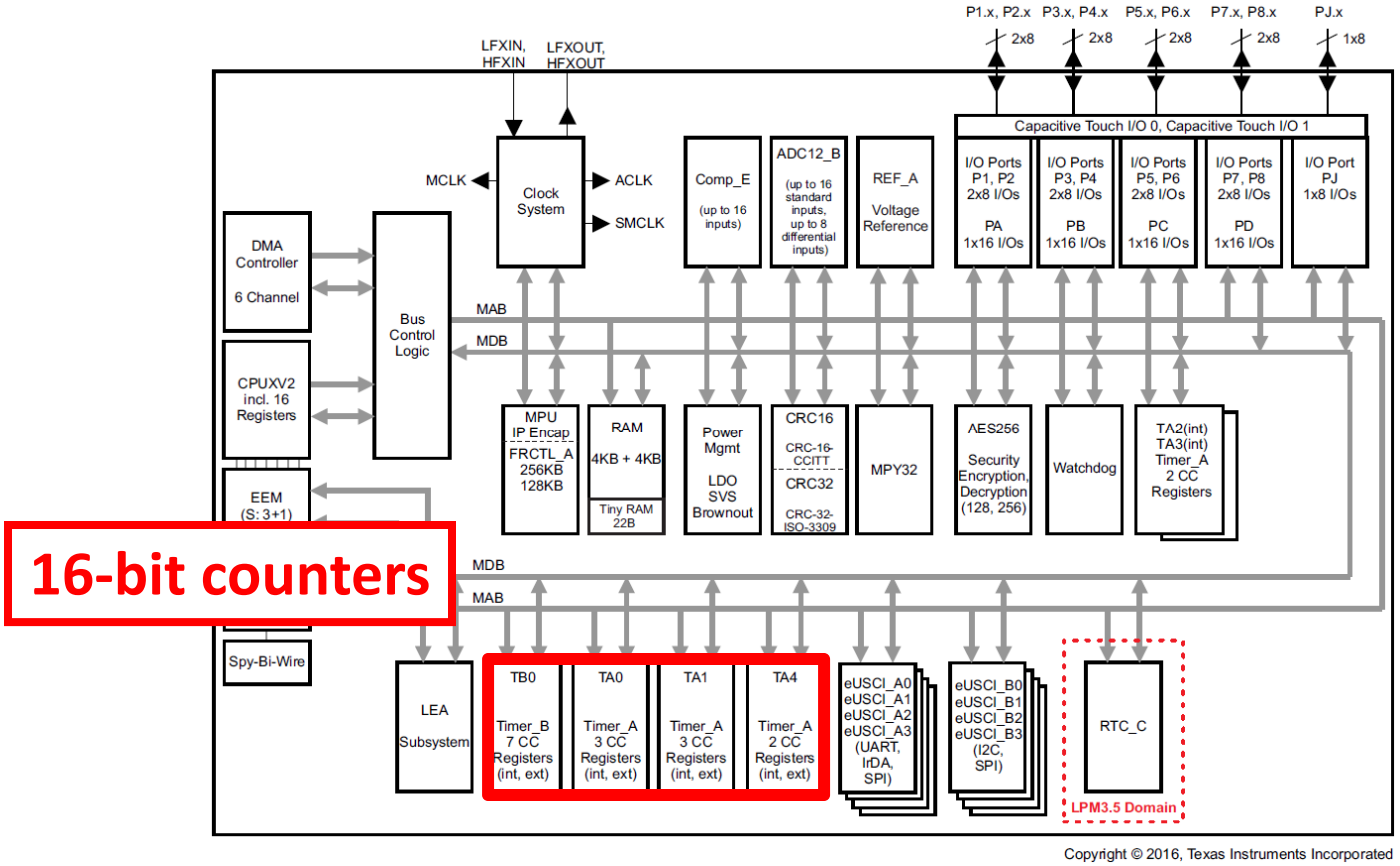
HAL Multiply
42 cycles


```
mymul:
UBA       #8,SP           // 2
.W        r13,6(SP)       // 4
V.W       r12,4(SP)       // 4
CALLA     #__mspabi_mpyul // ~100
MOV.W     r12,0(SP)       // 4
MOV.W     r13,2(SP)       // 4
ADDA      #8,SP           // 2
RETA                      // 2
```

SW Multiply
122 cycles

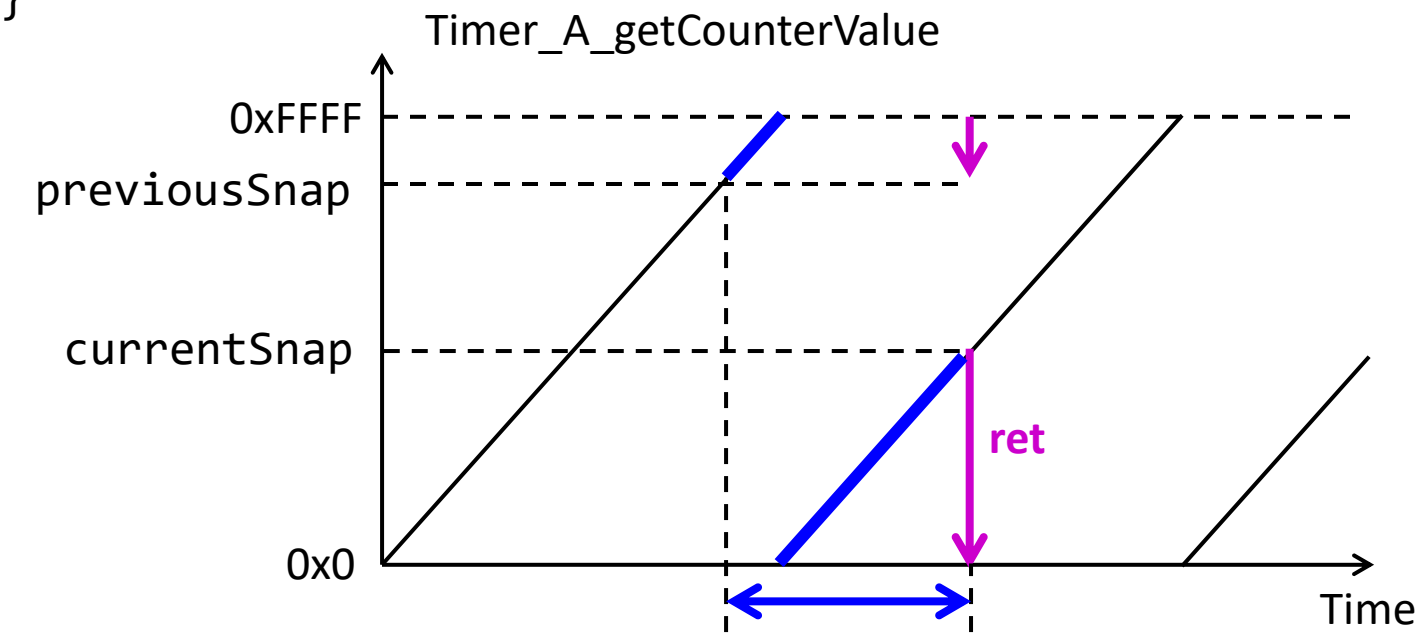
Intermezzo - Measuring Time

- **Measure performance as wall clock time**
- **Using a hardware counter**



Measuring Time

```
unsigned TimerLap() {
    static unsigned int previousSnap;
    unsigned int currentSnap, ret;
    currentSnap = Timer_A_getCounterValue(TIMER_A1_BASE);
    ret = (currentSnap - previousSnap);
    previousSnap = currentSnap;
    return ret;
}
```

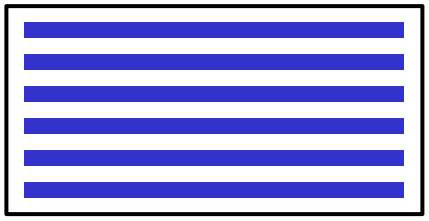


$t_{elapsed} = ret * T_{CLK}$

Measuring Time

```
unsigned TimerLap() {
    static unsigned int previousSnap;
    unsigned int currentSnap, ret;
    currentSnap = Timer_A_getCounterValue(TIMER_A1_BASE);
    ret = (currentSnap - previousSnap);
    previousSnap = currentSnap;
    return ret;
}
```

TimerLap();



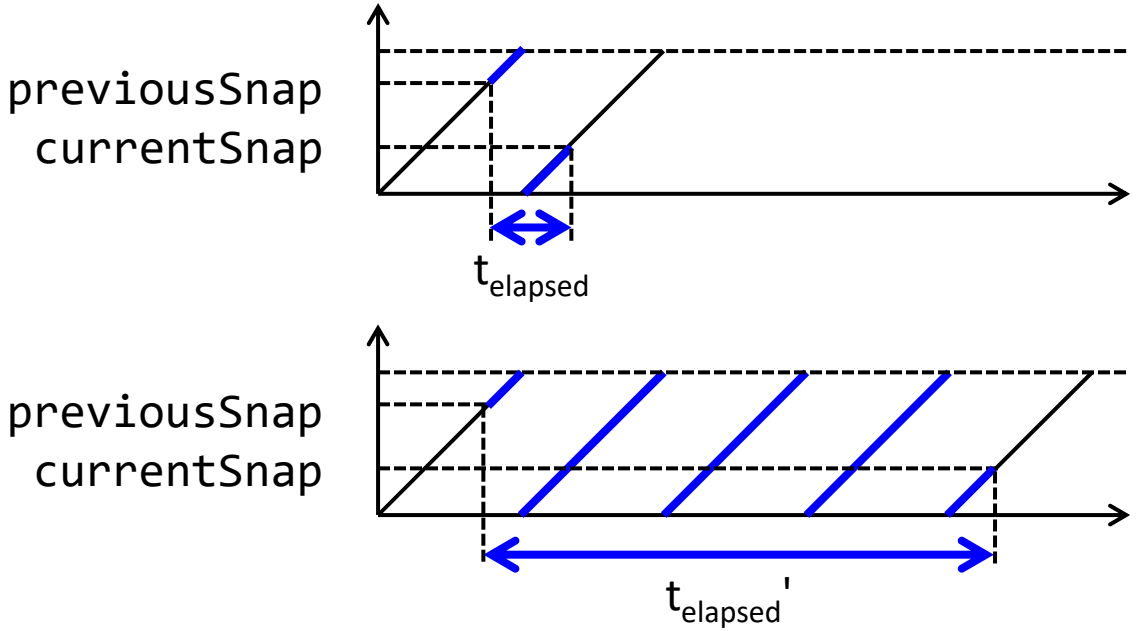
c = TimerLap();

Code to evaluate

1) What if Code is really long?
2) How precise is TimerLap()?

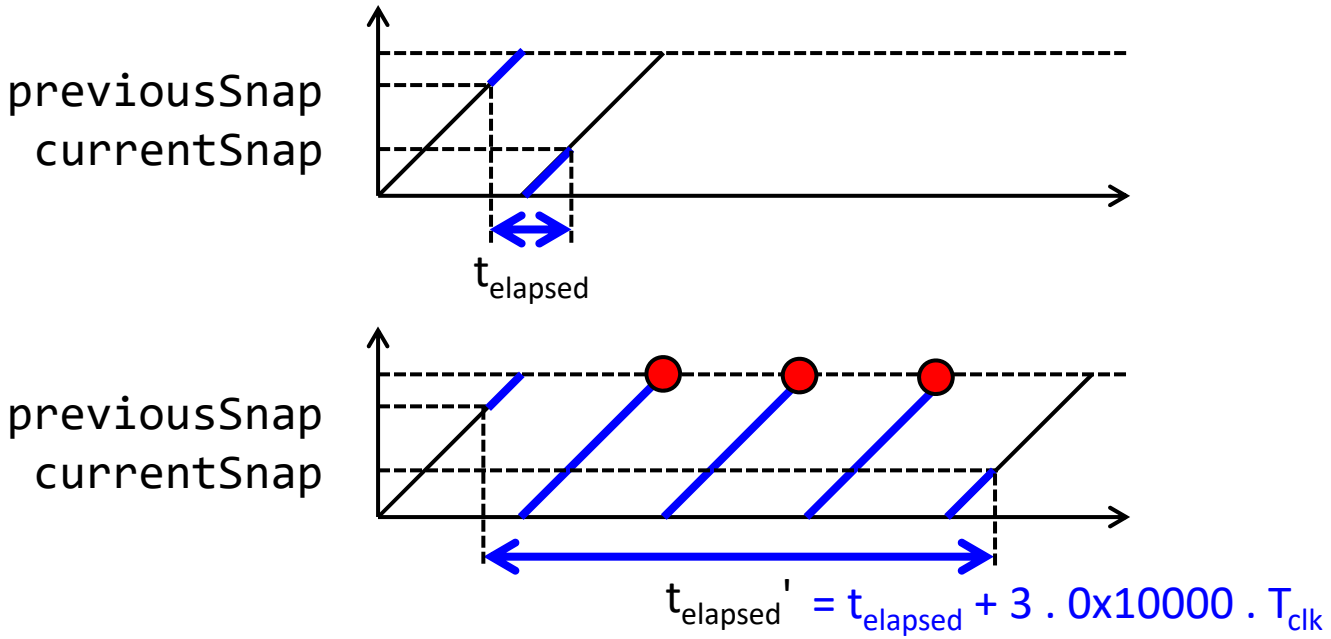
Measuring *Longer* Times ?

```
unsigned TimerLap() {
    static unsigned int previousSnap;
    unsigned int currentSnap, ret;
    currentSnap = Timer_A_getCounterValue(TIMER_A1_BASE);
    ret = (currentSnap - previousSnap);
    previousSnap = currentSnap;
    return ret;
}
```



Measuring *Longer* Times ?

```
unsigned TimerLap() {
    static unsigned int previousSnap;
    unsigned int currentSnap, ret;
    currentSnap = Timer_A_getCounterValue(TIMER_A1_BASE);
    ret = (currentSnap - previousSnap);
    previousSnap = currentSnap;
    return ret;
}
```



Measuring *Longer* Times ?

```
static uint32_t currentInt;
uint32_t TimerLap() {
    static unsigned int previousSnap;
    unsigned currentSnap;
    uint32_t ret;
    Timer_A_disableInterrupt(TIMER_A1_BASE);
    currentSnap = Timer_A_getCounterValue(TIMER_A1_BASE);
    if (currentSnap < previousSnap)
       ret = (uint16_t) (currentSnap-previousSnap)+(currentInt-1) << 16;
    else
       ret = (uint16_t) (currentSnap-previousSnap) + currentInt << 16;
    currentInt = 0;
    previousSnap = currentSnap;
    Timer_A_enableInterrupt(TIMER_A1_BASE);
    return ret;
}
#pragma vector=TIMER1_A1_VECTOR
__interrupt void TIMER1_A1_ISR (void) {
   if (TA1IV == 14) // overflow
      currentInt = currentInt + 1;
}
```

Overhead

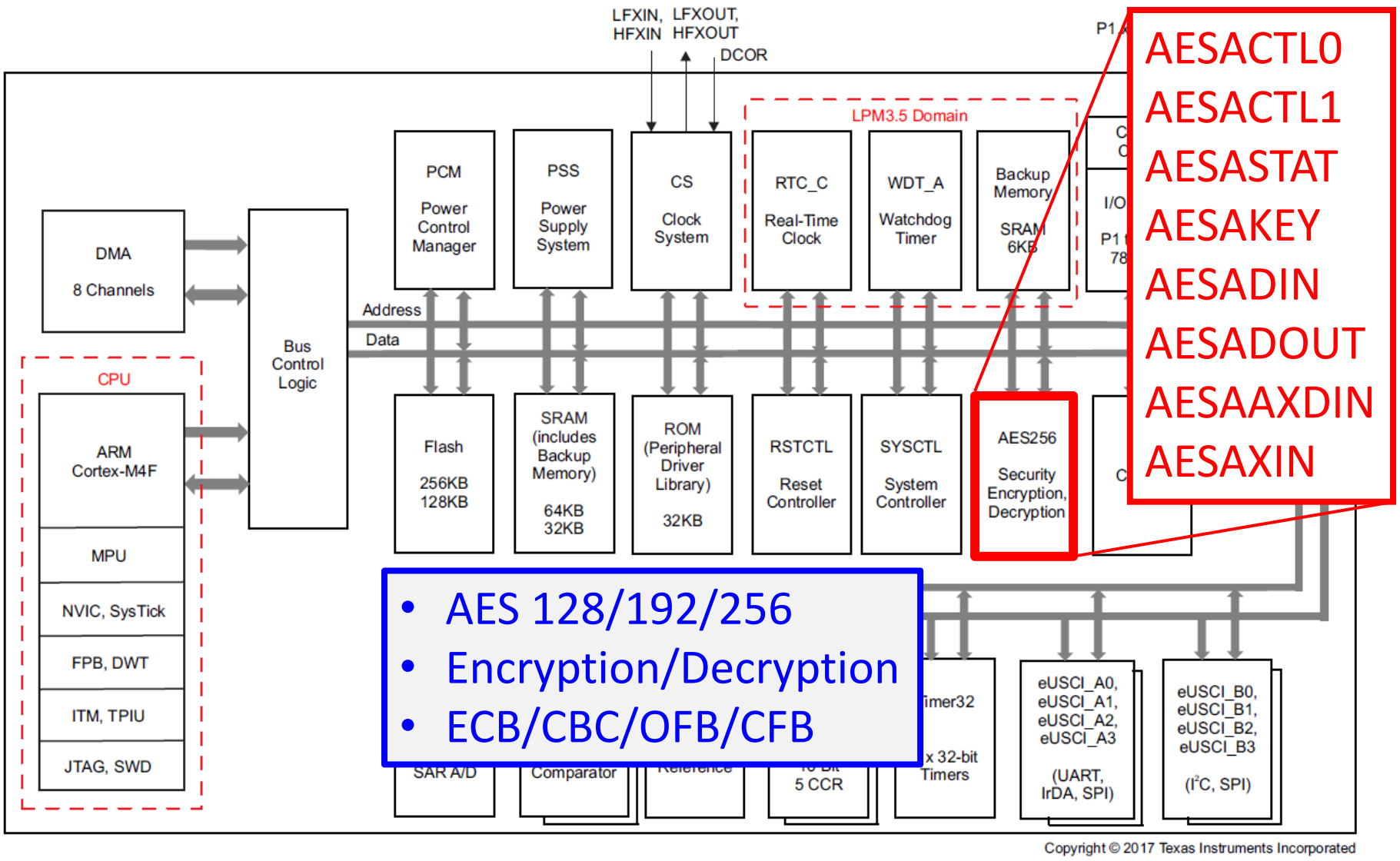
- **MSP430**

```
uint32_t c;
TimerLap();
c = TimerLap();  // 160 cycles + Timer_interrupts . K
```

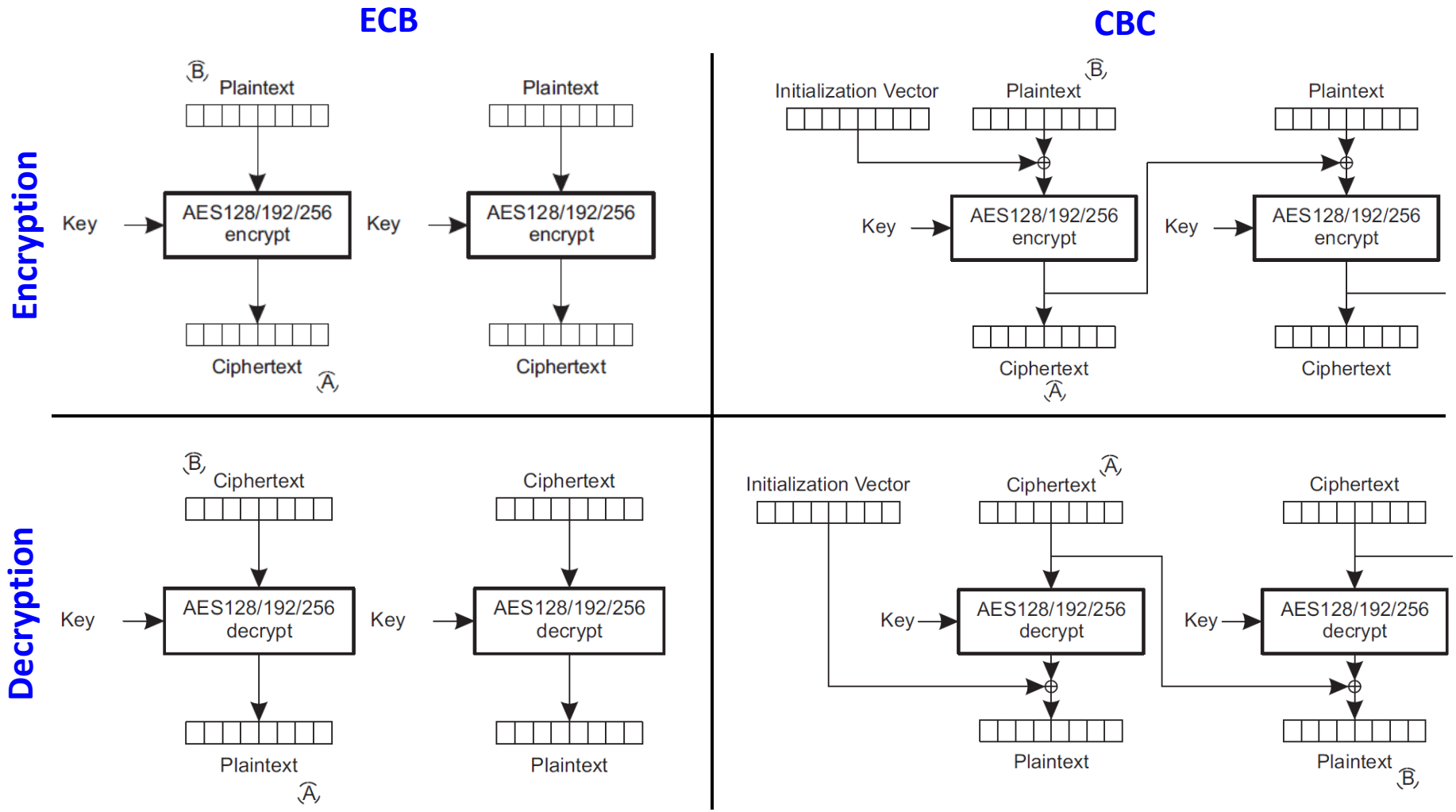
Outline

1. Fundamentals of Parallelism
2. Embedded Architecture of MSP430, MSP432
3. Hardware Acceleration in Embedded Architectures
4. AES Hardware Accelerator
5. Direct Memory Access
6. Power Dissipation
7. Literature

AES Acceleration

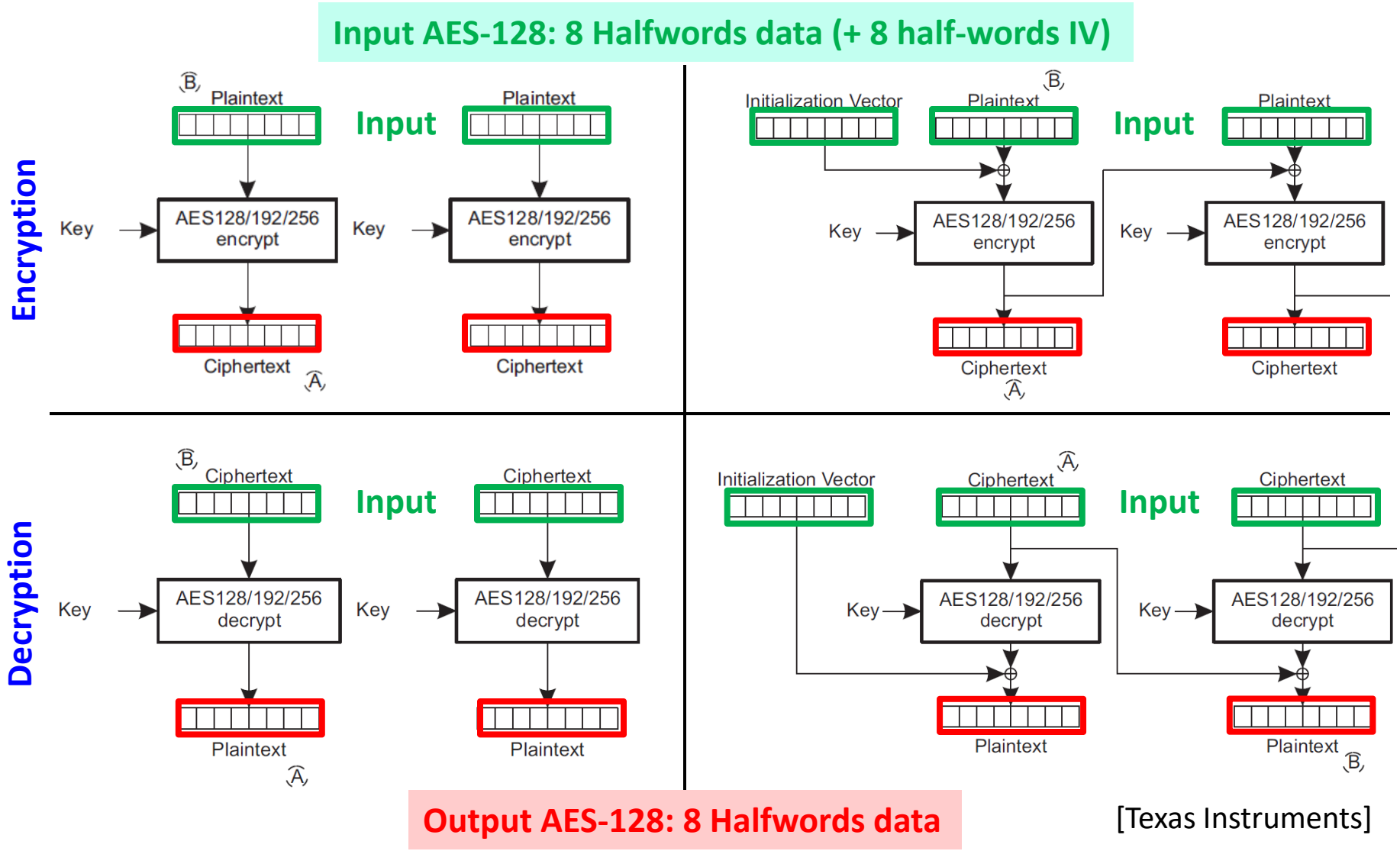


ECB, CBC



[Texas Instruments]

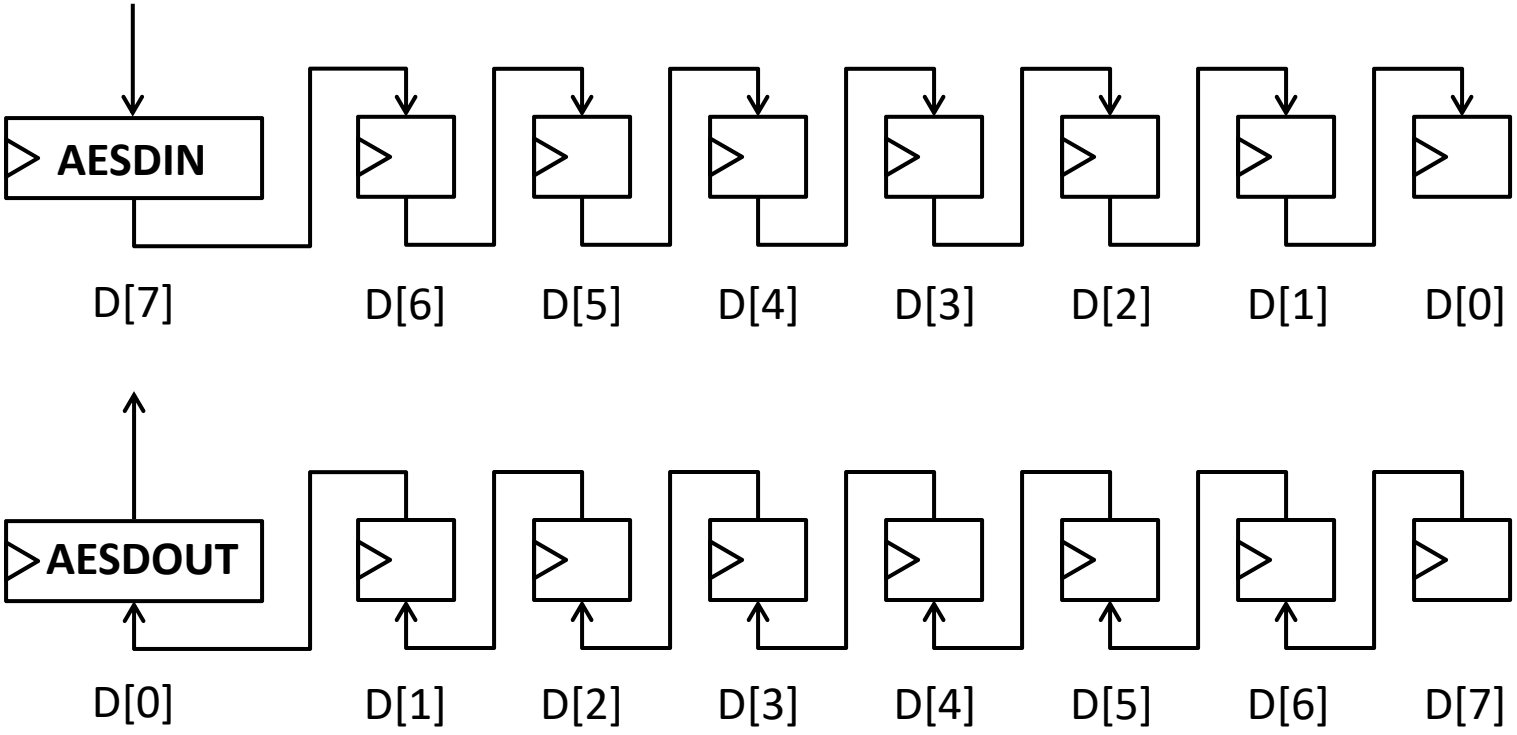
ECB, CBC



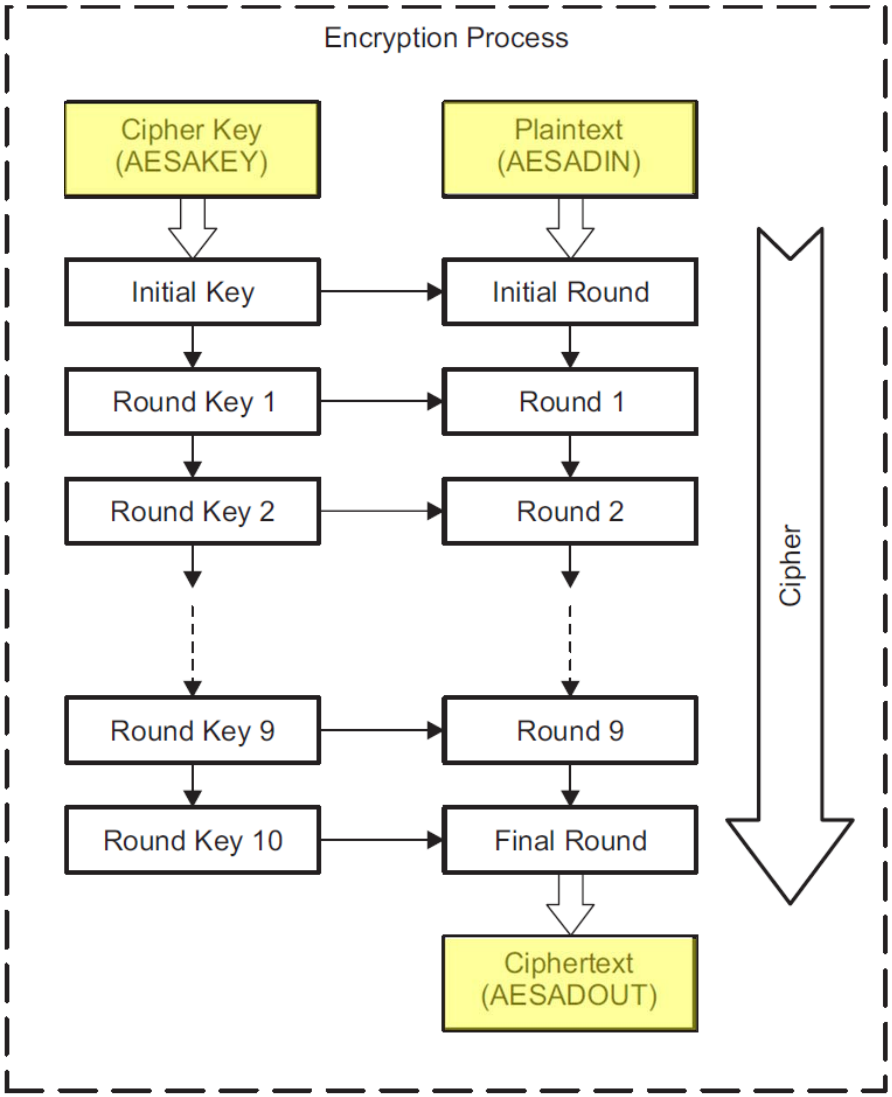
[Texas Instruments]

Input/Output Register

- **One memory-mapped location sequentially loads plaintext (or ciphertext, or key, or IV)**

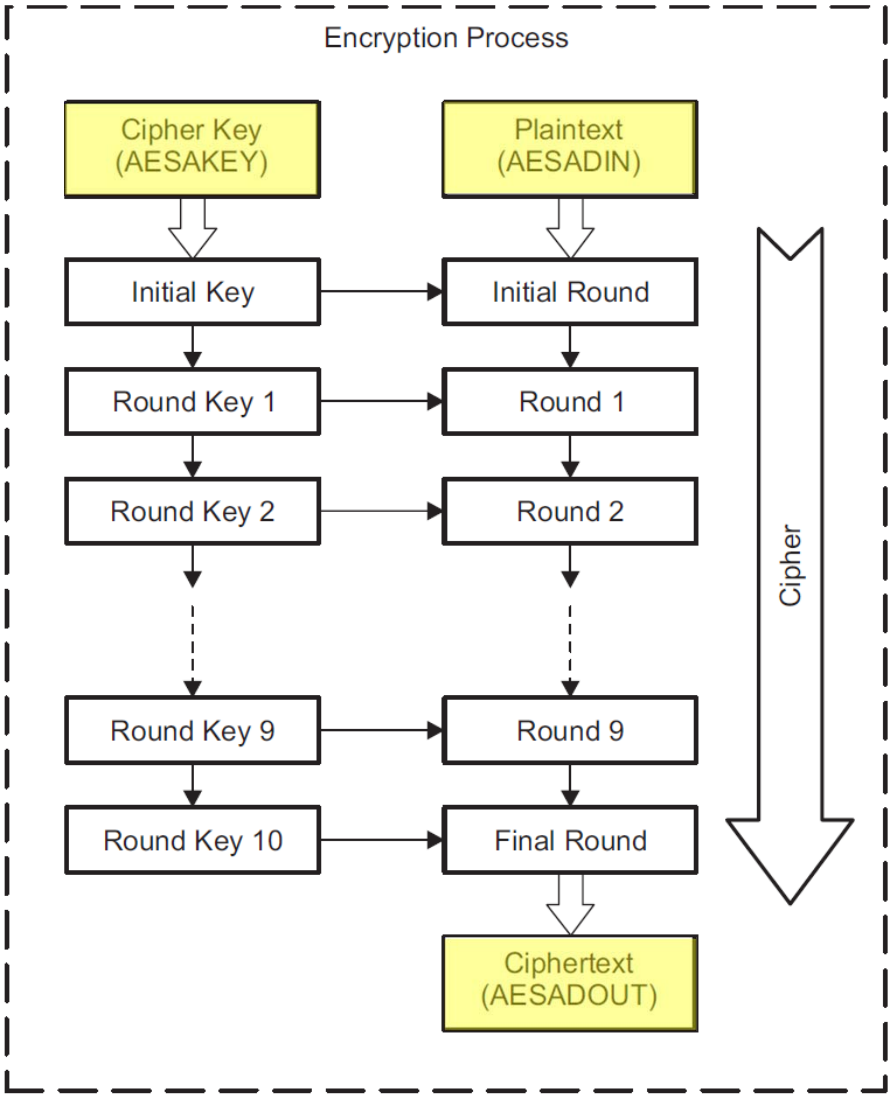


Key Schedule and Encryption



[Texas Instruments]

Key Schedule and Encryption



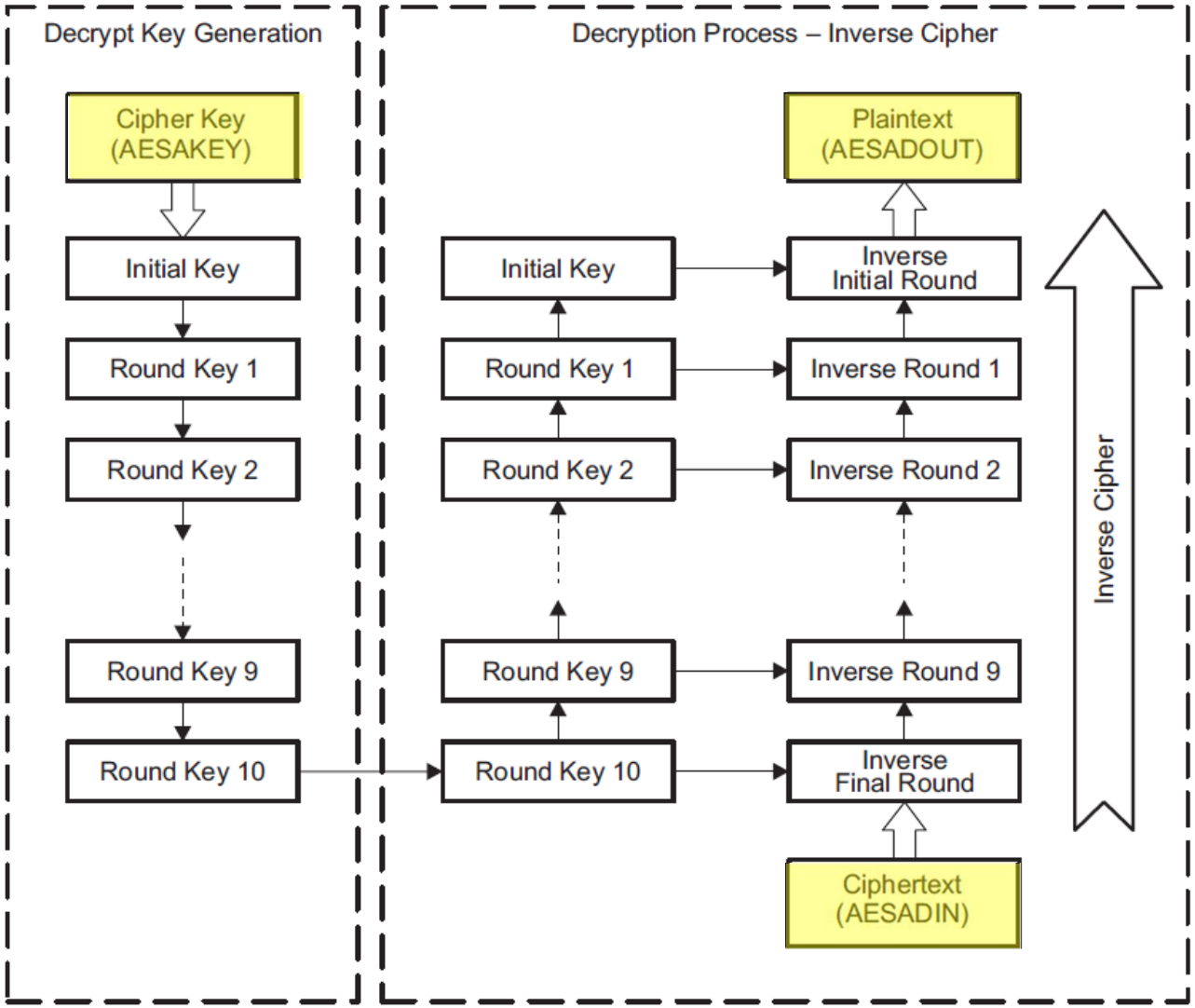
[Texas Instruments]

168 Cycles

Minimum Area Design
10 Rounds
16 Sbox/Round

Encryption Mode

Online Key Schedule and Decryption

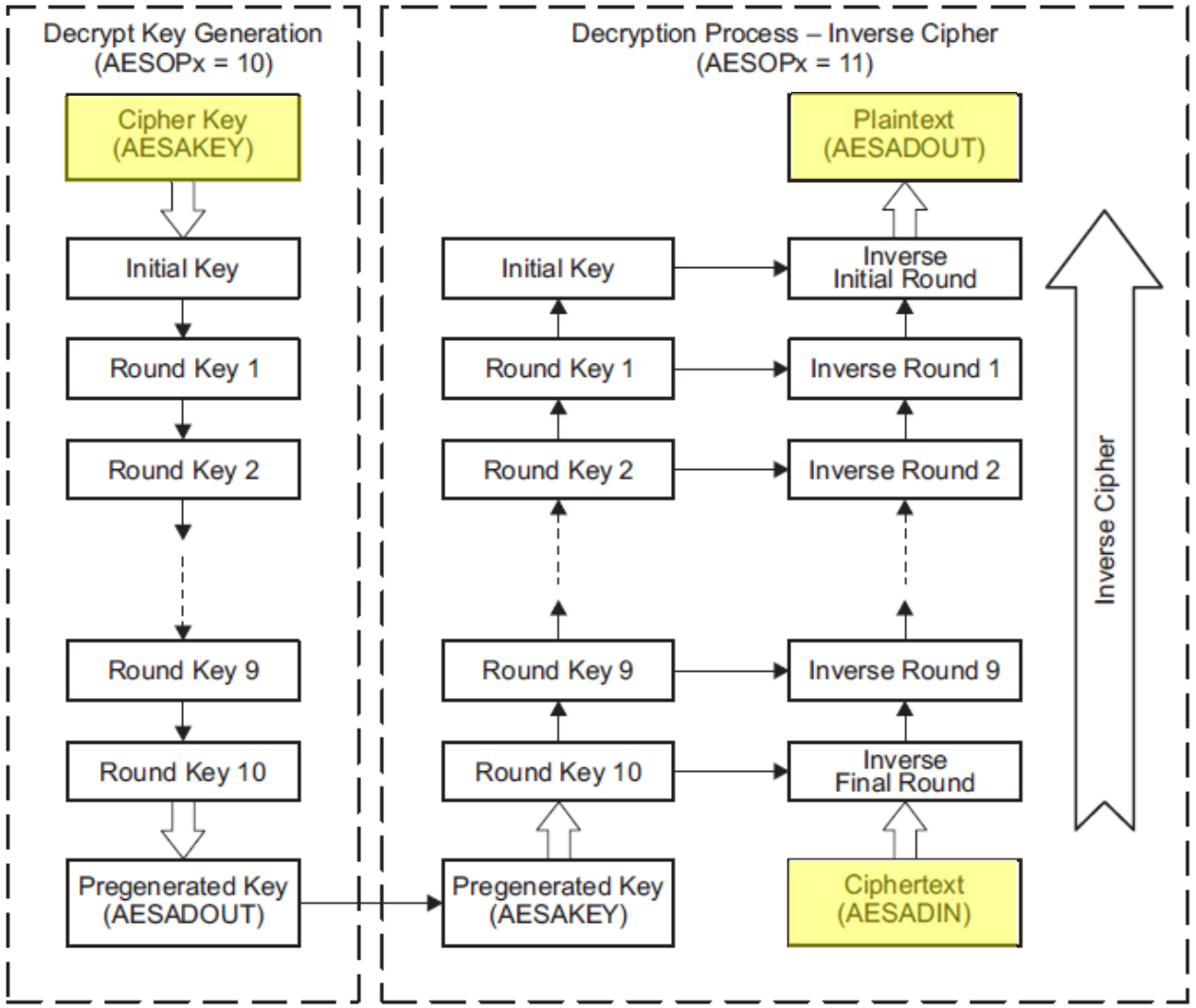


215 Cycles

Decryption with Initial Roundkey

[Texas Instruments]

Offline Key Schedule and Decryption



[Texas Instruments]

53 Cycles Key Sched

+

168 Cycles Decryption

Encryption Key Schedule

+

Decryption with Last Roundkey

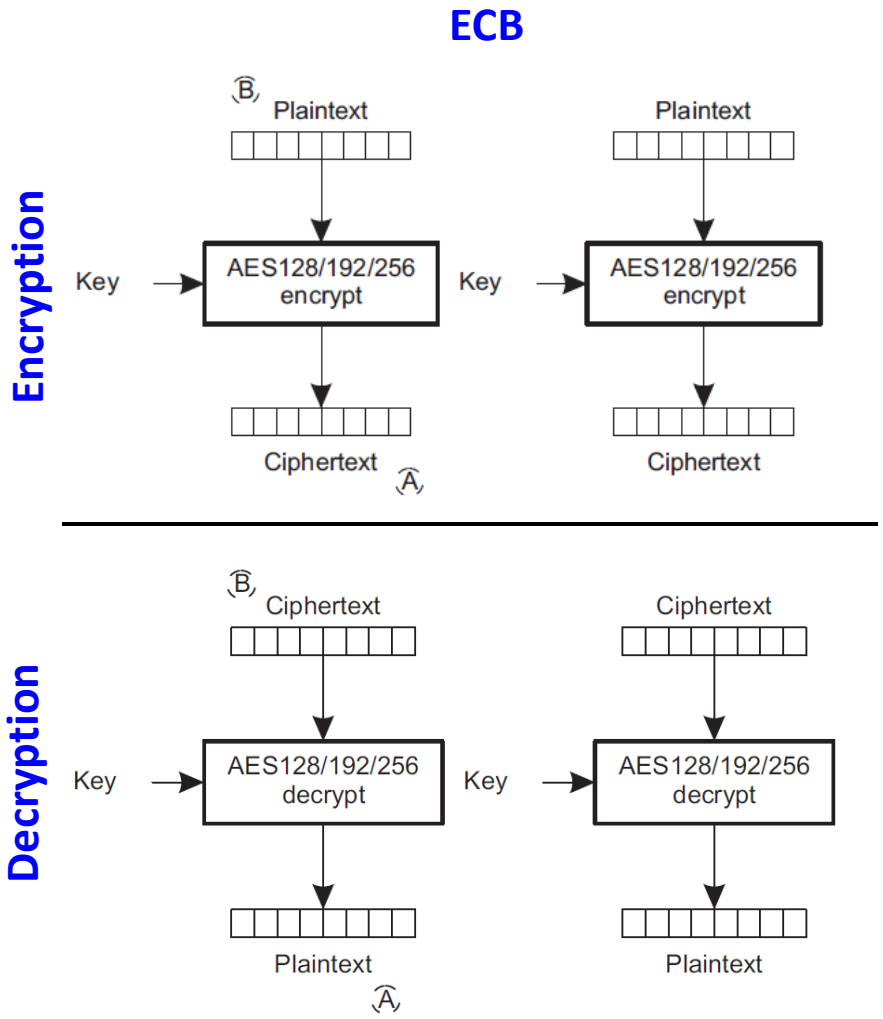
Reference Implementation

Design	Reference MSP430	Reference MSP432	Stoffelen al. M4 [SAC16]
AES Key Schedule	5,861	1,683	294.8
AES Enc ECB	12,831	4,384	661.7
AES Dec ECB	27,753	13,127	648.3
AES Enc CBC	12,268	4,634	
AES Dec CBC	28,458	13,277	

Max optimization level (speed)
Per-block cycles counts in
1-block ECB benchmark,
4-block CBC benchmark

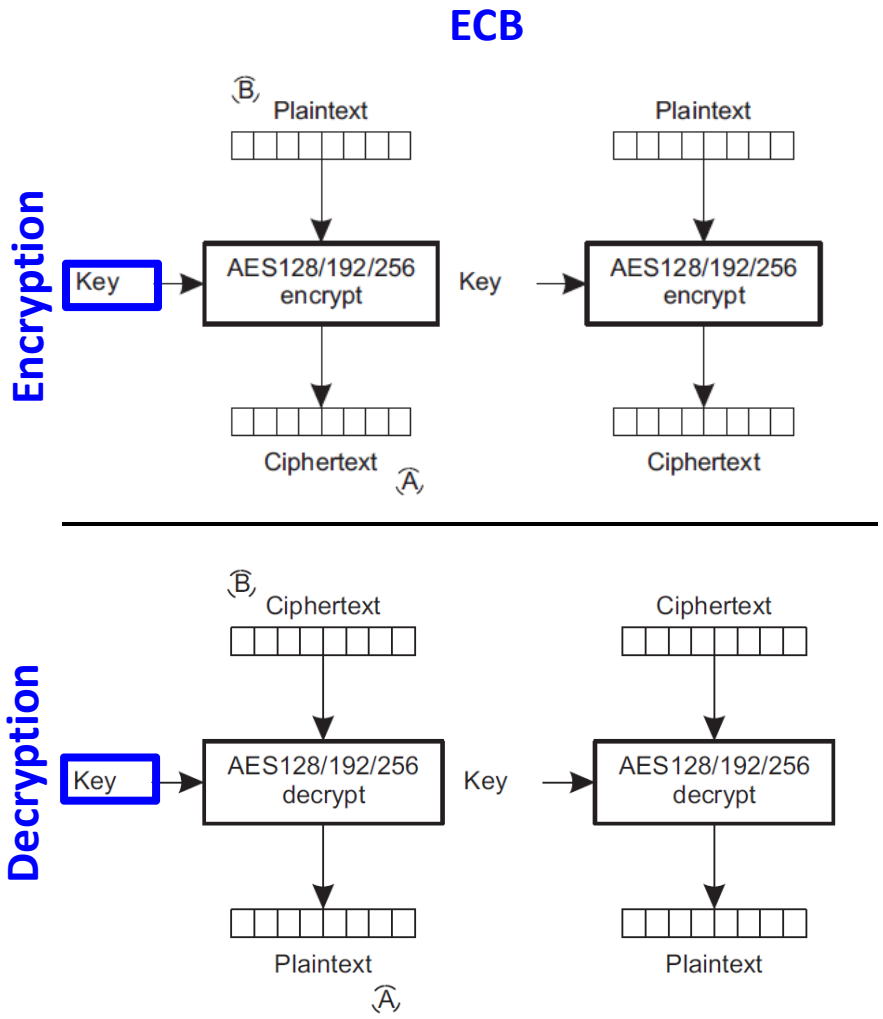
https://github.com/kokke/tiny-AES-c

ECB operation



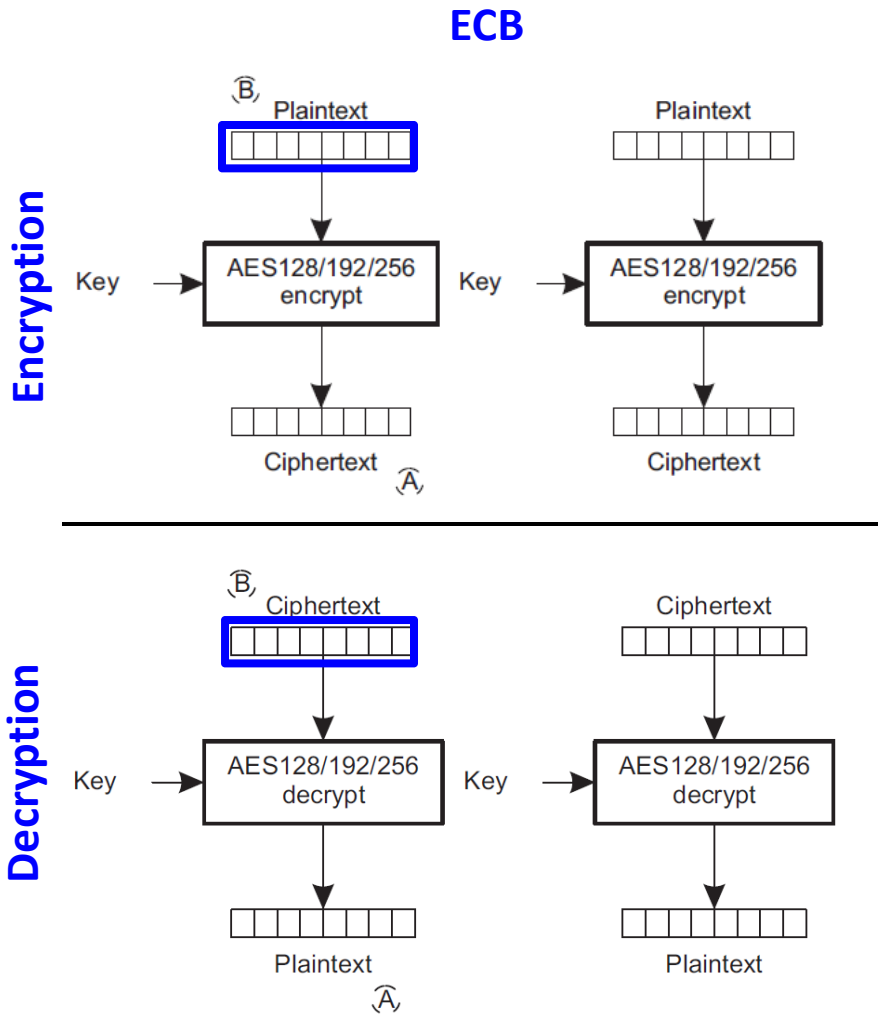
1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Ciphertext**

ECB operation



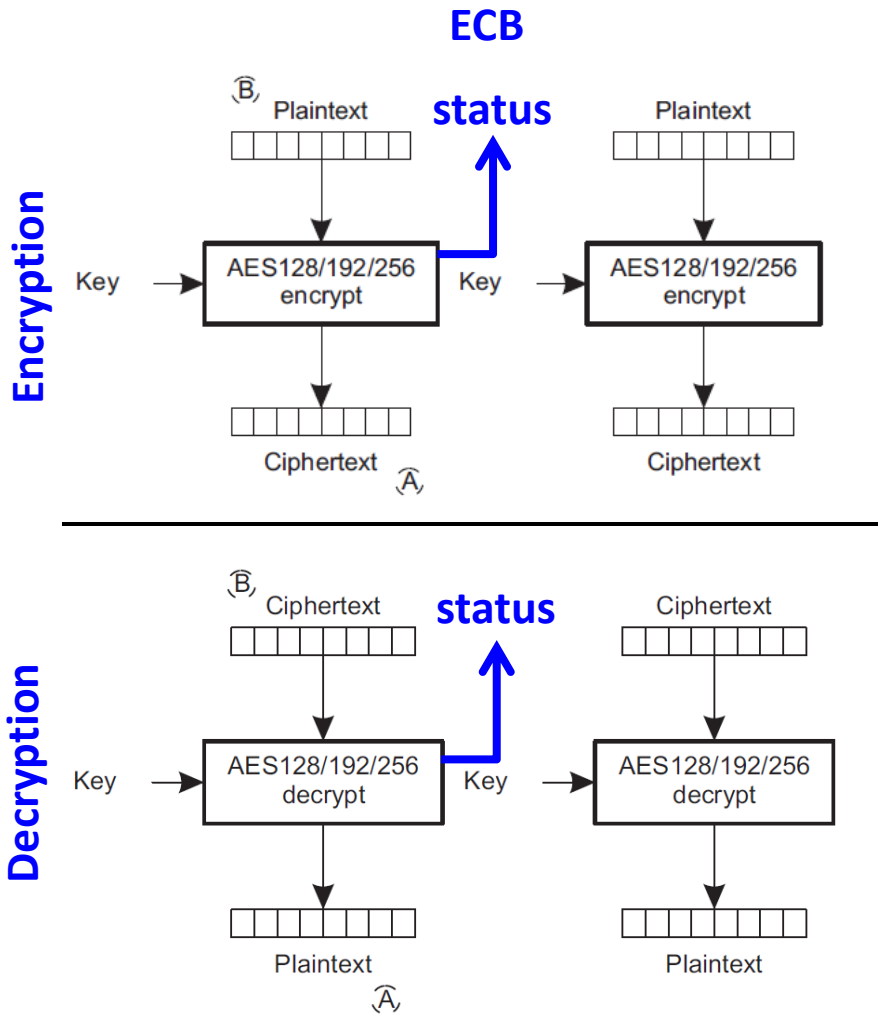
1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Ciphertext**

ECB operation



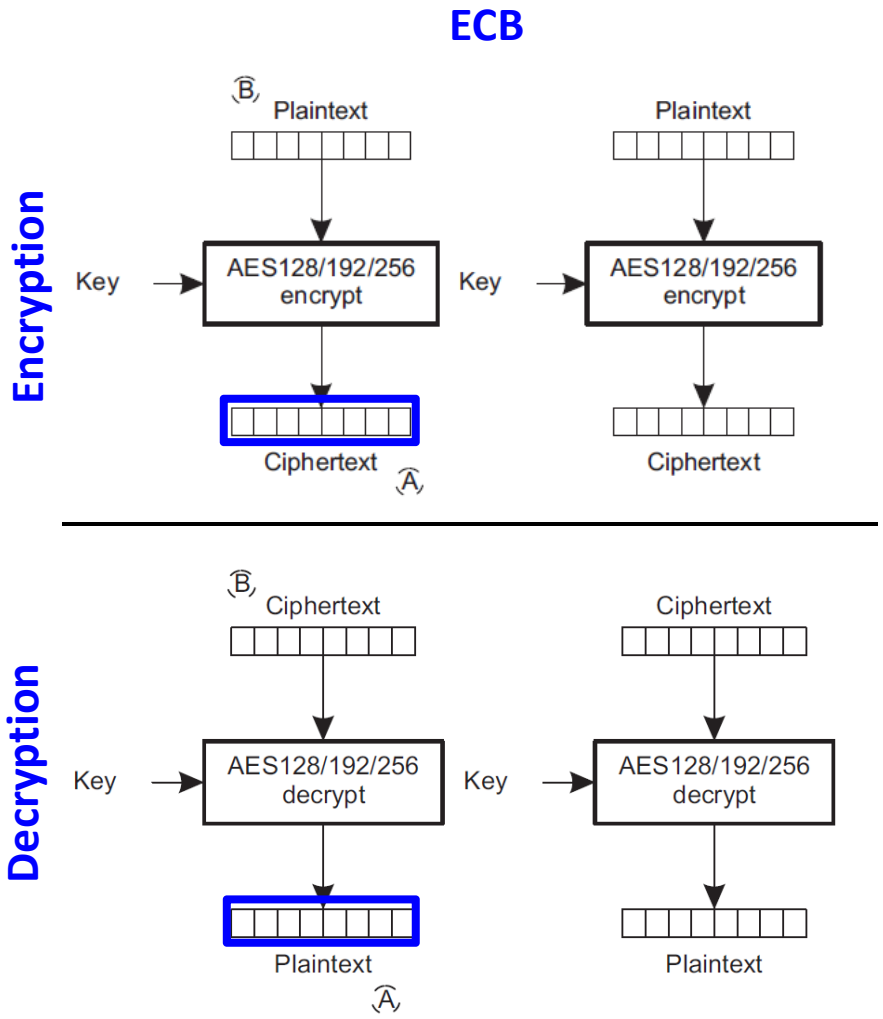
1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Ciphertext**

ECB operation



1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Ciphertext**

ECB operation



1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Cipher/Plaintext**

ECB Encryption (MSP430)

```
#include "msp430.h"

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;

    AESACTL0 = AESSWRST;
    AESACTL0 = (AESACTL0 & ~AESOP) | AESOP_0;
    AESACTL0 = (AESACTL0 & ~AESKL) | AESKL_0;

    uint8_t i;
    for (i=0; i<8; i++)
        AESAKEY = ((uint16_t *) CipherKey)[i];

    uint16_t k;
    for (k=0; k<32; k++) {
        for (i=0; i<8; i++)
            AESADIN = ((uint16_t *) Data)[i];
        while (AESASTAT & AESBUSY) ;
        for (i=0; i<8; i++)
            ((uint16_t *) DataAESencrypted)[i] = AESADOUT;
    }
}
```

ECB Encryption (MSP430)

```
#include "msp430.h"

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;

    AESACTL0 = AESSWRST;
    AESACTL0 = (AESACTL0 & ~AESOP) |
    AESACTL0 = (AESACTL0 & ~AESKL) |

    uint8_t i;
    for (i=0; i<8; i++)
        AESAKEY = ((uint16_t *) Ciph

    uint16_t k;
    for (k=0; k<32; k++) {
        for (i=0; i<8; i++)
            AESADIN = ((uint16_t *)
        while (AESASTAT & AESBUSY) ;
        for (i=0; i<8; i++)
            ((uint16_t *) DataAESencrypted)[i] = AESADOUT;
    }
}
```

(msp430fr5994.h)

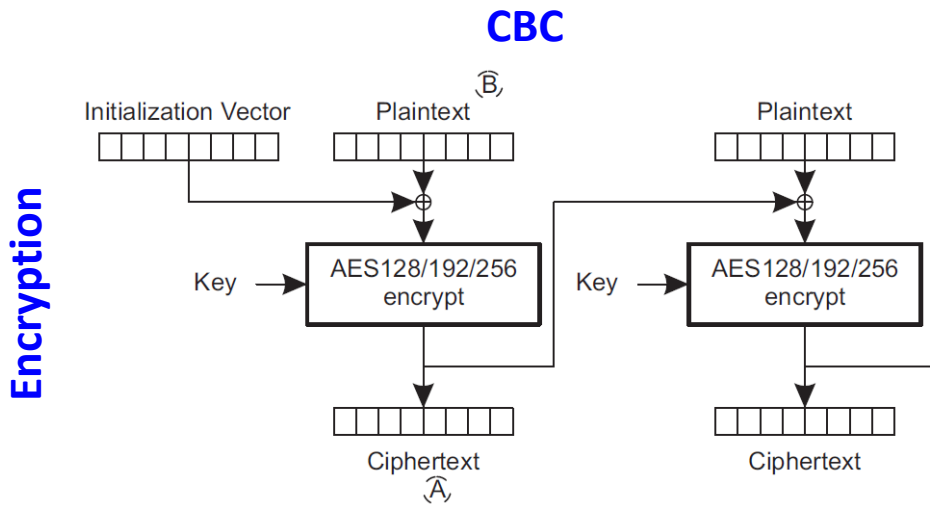
```
extern volatile unsigned AESACTTL0;
extern volatile unsigned AESAKEY;
extern volatile unsigned AESADIN;
extern volatile unsigned AESADOUT;
```

Resolved by the linker

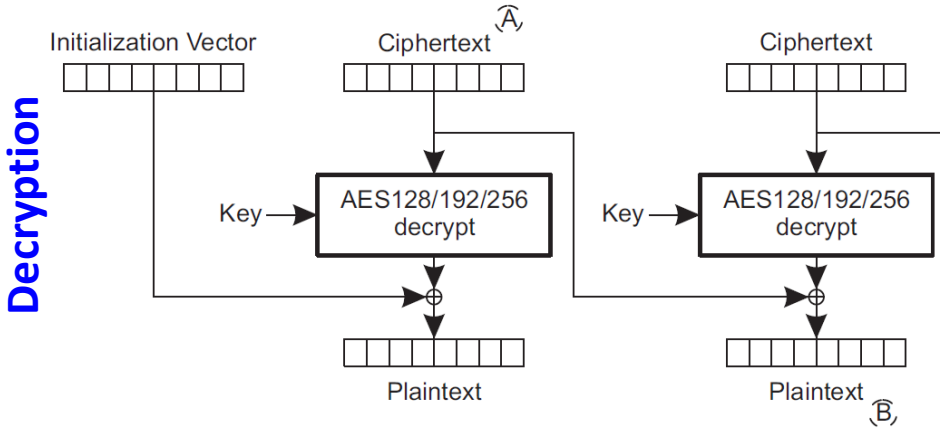
(msp430fr5994.cmd)

```
AESACTL0        = 0x09C0;
AESASTAT        = 0x09C4;
AESAKEY         = 0x09C6;
AESADIN         = 0x09C8;
AESADOUT        = 0x09CA;
```

CBC Operation



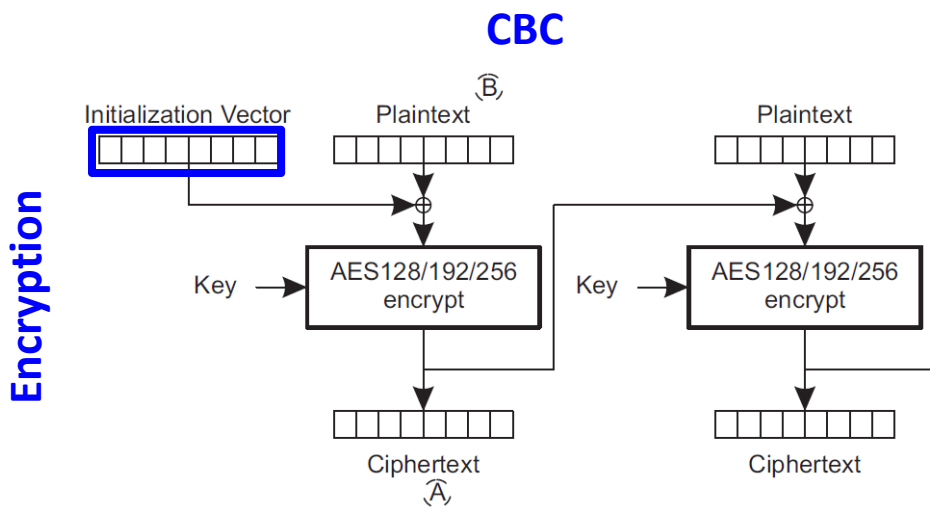
1. **Set mode, keylength**
2. **Load Key, IV**
3. **For each block**
   a. **Load Block**
   b. **Wait until complete**
   c. **Read Ciphertext**



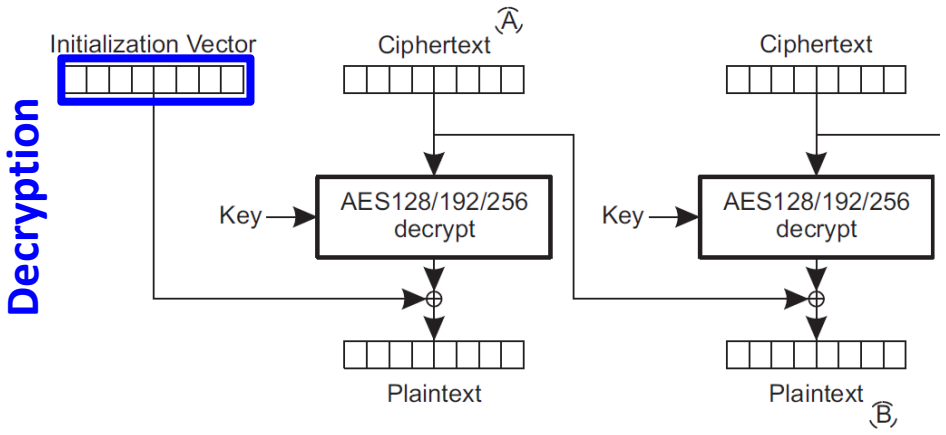
[Texas Instruments]

1. **Set mode, keylength**
2. **Load Key**
3. **For each block**
   a. **Load IV or previous Ciphertext**
   b. **Load Block**
   c. **Wait until complete**
   d. **Read Plaintext**

CBC Operation



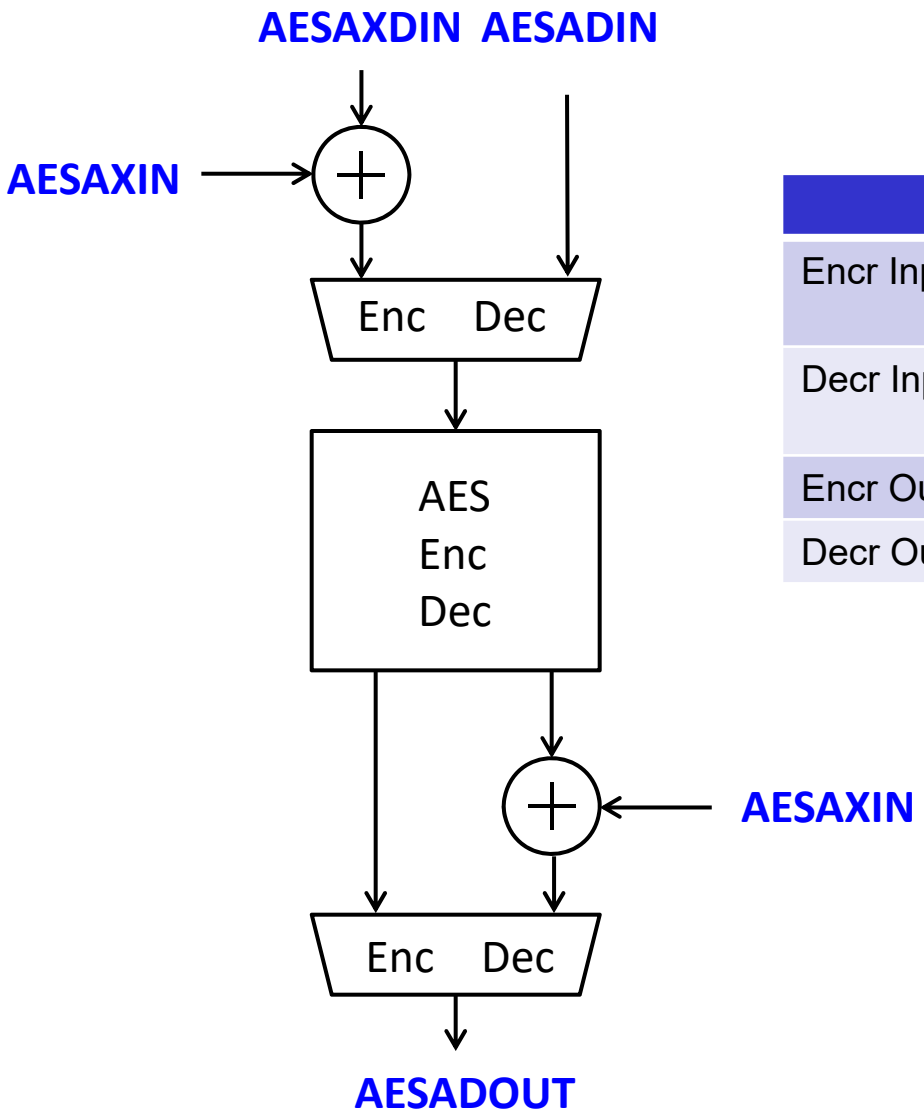
1. Set mode, keylength
2. Load Key, IV
3. For each block
   a. Load Block
   b. Wait until complete
   c. Read Ciphertext



[Texas Instruments]

1. Set mode, keylength
2. Load Key
3. For each block
   a. Load IV or previous Ciphertext
   b. Load Block
   c. Wait until complete
   d. Read Plaintext

I/O in CBC Operation



	ECB	CBC
Encr Input	AESADIN	AESAXDIN AESAXIN
Decr Input		AESADIN AESAXIN
Encr Output	AESADOUT	
Decr Output		

CBC Decryption (MSP430)

```
AESACTL0 = AESSWRST;
AESACTL0 = (AESACTL0 & ~AESKL) | AESKL_0;
AESACTL0 = (AESACTL0 & ~AESOP) | AESOP_2; // key schedule
for (i=0; i<8; i++) AESAKEY = ((uint16_t *) CipherKey)[i];
while (AESASTAT & AESBUSY) ;
AESACTL0 = (AESACTL0 & ~AESOP) | AESOP_3; // decryption with offline keysch
AESASTAT |= AESKEYWR;

// decrypt 4 blocks
for (k=0; k<4; k++) {
  for (i=0; i<8; i++)
    AESAXIN = k ? ((uint16_t *) DataAESencrypted)[i+(k-1)*8] :
                  ((uint16_t *) IV)[i];
  for (i=0; i<8; i++)
    AESADIN = ((uint16_t *) DataAESencrypted)[i + k * 8];

  while (AESASTAT & AESBUSY) ;
  for (i=0; i<8; i++)
      ((uint16_t *) DataAESdecrypted)[i + k * 8] = AESADOUT;

}
```

Performance of AES Accelerator

Design	Reference MSP430	Reference MSP432	Schwabe al. M4
AES Key Schedule	5,861	1,683	294.8
AES Enc ECB	12,831	4,384	661.7
AES Dec ECB	27,753	13,127	648.3
AES Enc CBC	12,268	4,634	
AES Dec CBC	28,458	13,277	

AES Key Schedule	52	55
AES Enc ECB	250	450
AES Dec ECB	250	548
AES Enc CBC	282	452
AES Dec CBC	423	622

Max optimization level (speed)
Per-block cycles counts
benchmarking 1K-block (MSP 432) or 32 block (MSP430)

Performance of AES Accelerator

Design	Reference MSP430	Reference MSP432	Schwabe al. M4
AES Key Schedule	5,861	1,683	294.8
AES Enc ECB	**12,831**	**4,384**	661.7
AES Dec ECB	27,753	13,127	648.3
AES Enc CBC	12,268	4,634	
AES Dec CBC			

51x

9.7x

AES Key Schedule	52	55
AES Enc ECB	**250**	**450**
AES Dec ECB	250	548
AES Enc CBC	282	452
AES Dec CBC	423	622

Max optimization level (speed)
Per-block cycles counts
benchmarking 1K-block (MSP 432) or 32 block (MSP430)

Performance of AES Accelerator

Design	Reference MSP430	Reference MSP432	Schwabe al. M4
AES Key Schedule	5,861	1,683	294.8
AES Enc ECB	**12,831**	**4,384**	661.7
AES Dec ECB	27,753	13,127	648.3
AES Enc CBC	12,268	4,634	
AES Dec CBC			

AES Key Schedule	52	55
AES Enc ECB	**250**	**450**
AES Dec ECB	250	548
AES Enc CBC		
AES Dec CBC		

51x

9.7x

82 cycles overhead

282 cycles overhead

Max optimization level (speed)

Pe

168 cycles ECB Encr

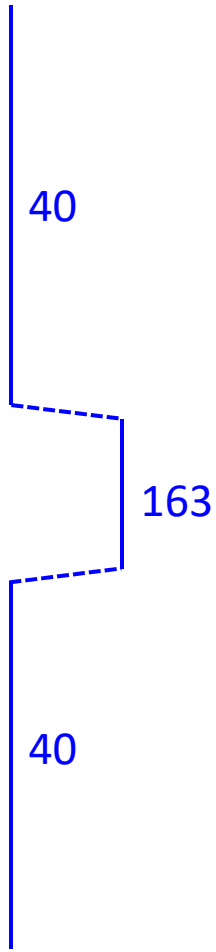
benchmarking 1K-block (MSP 432) or 32 block (MSP430)

Where do the cycles go? C

```
uint16_t k;
for (k=0; k<32; k++) {
  for (i=0; i<8; i++)
    AESADIN = ((uint16_t *) Data)[i];

  while (AESASTAT & AESBUSY) ;

  for (i=0; i<8; i++)
    ((uint16_t *) DataAESencrypted)[i] = AESADOUT;
}
```

Where do the cycles go? MSP430

```
        MOV.W   #32,r15                         // 2
$C$L2:                                                  SW   HW
        MOV.W   &Data+0,&AESADIN+0              // 5
        MOV.W   &Data+2,&AESADIN+0              // 5
        MOV.W   &Data+4,&AESADIN+0              // 5
        MOV.W   &Data+6,&AESADIN+0              // 5
        MOV.W   &Data+8,&AESADIN+0              // 5
        MOV.W   &Data+10,&AESADIN+0             // 5
        MOV.W   &Data+12,&AESADIN+0             // 5
        MOV.W   &Data+14,&AESADIN+0             // 5
$C$L3:
        BIT.W   #1,&AESASTAT+0                  // 5
        JNE     $C$L3                           // 2
        MOV.W   &AESADOUT+0,&DataAESencrypted+0 // 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+2 // 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+4 // 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+6 // 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+8 // 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+10// 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+12// 5
        MOV.W   &AESADOUT+0,&DataAESencrypted+14// 5
        SUB.W   #1,r15                          // 2
        JNE     $C$L2                           // 2
```



Where do the cycles go? MSP432

```
 MOVS      A1, #4
||$C$L9||:
        LDRH      A2, [SP, #0]
        STRH      A2, [V1, #8]
        LDRH      LR, [SP, #2]
        STRH      LR, [V1, #8]
        LDRH      V9, [SP, #4]
        STRH      V9, [V1, #8]
        LDRH      V6, [SP, #6]
        STRH      V6, [V1, #8]
        LDRH      A4, [SP, #8]
        STRH      A4, [V1, #8]
        LDRH      A3, [SP, #10]
        STRH      A3, [V1, #8]
        LDRH      LR, [SP, #12]
        STRH      LR, [V1, #8]
        LDRH      V9, [SP, #14]
        STRH      V9, [V1, #8]
        LDRH      A2, [V1, #4]
        LSRS      A2, A2, #1
        BCC       ||$C$L11||
```

```
||$C$L10||:
        LDRH      A2, [V1, #4]
        LSRS      A2, A2, #1
        BCS       ||$C$L10||
||$C$L11||:
        LDRH      A3, [V1, #10]
        STRH      A3, [SP, #144]
        LDRH      A2, [V1, #10]
        STRH      A2, [SP, #146]
        LDRH      A3, [V1, #10]
        STRH      A3, [SP, #148]
        LDRH      A2, [V1, #10]
        STRH      A2, [SP, #150]
        LDRH      A3, [V1, #10]
        STRH      A3, [SP, #152]
        LDRH      A2, [V1, #10]
        STRH      A2, [SP, #154]
        LDRH      A3, [V1, #10]
        LDRH      A2, [V1, #10]
        STRH      A3, [SP, #156]
        SUBS      A1, A1, #1
        STRH      A2, [SP, #158]
        BNE       ||$C$L9||
```

Where do the cycles go? MSP432

```
 MOVS       A1, #4
||$C$L9||:
        LDRH       A2, [SP, #0]
        STRH       A2, [V1, #8]
        LDRH       LR, [SP, #2]
        STRH       V9, [V1, #8]
        LDRH       A2, [V1, #4]
        LSRS       A2, A2, #1
        BCC        ||$C$L11||
```

```
||$C$L10||:
        LDRH       A2, [V1, #4]
        LSRS       A2, A2, #1
        BCS        ||$C$L10||
||$C$L11||:
        LDRH       A3, [V1, #10]
        LDRH       A2, [V1, #10]
        STRH       A3, [SP, #156]
        SUBS       A1, A1, #1
        STRH       A2, [SP, #158]
        BNE        ||$C$L9||
```

It's not obvious to explain 282 cycles of overhead, but note:

- ARM Cortex M4 is a 32-bit architecture, but accelerator made for 16-bit architecture
- Load-store architecture doubles instruction count
- There is additional bus latency

Table 5-3. Peripheral Register Access Latency

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		MIN	MAX	UNIT
t_{reg_access}	Number of CPU clock cycles required for read or write access to peripheral registers	2 (1)	5 (2)	MCLK cycles

(1) The bridge that connects CPU to peripherals runs at half of the speed of the CPU.
(2) The maximum value depends on the previous opcode executing in the CPU pipeline and the status of the bus (idle or busy performing data access).

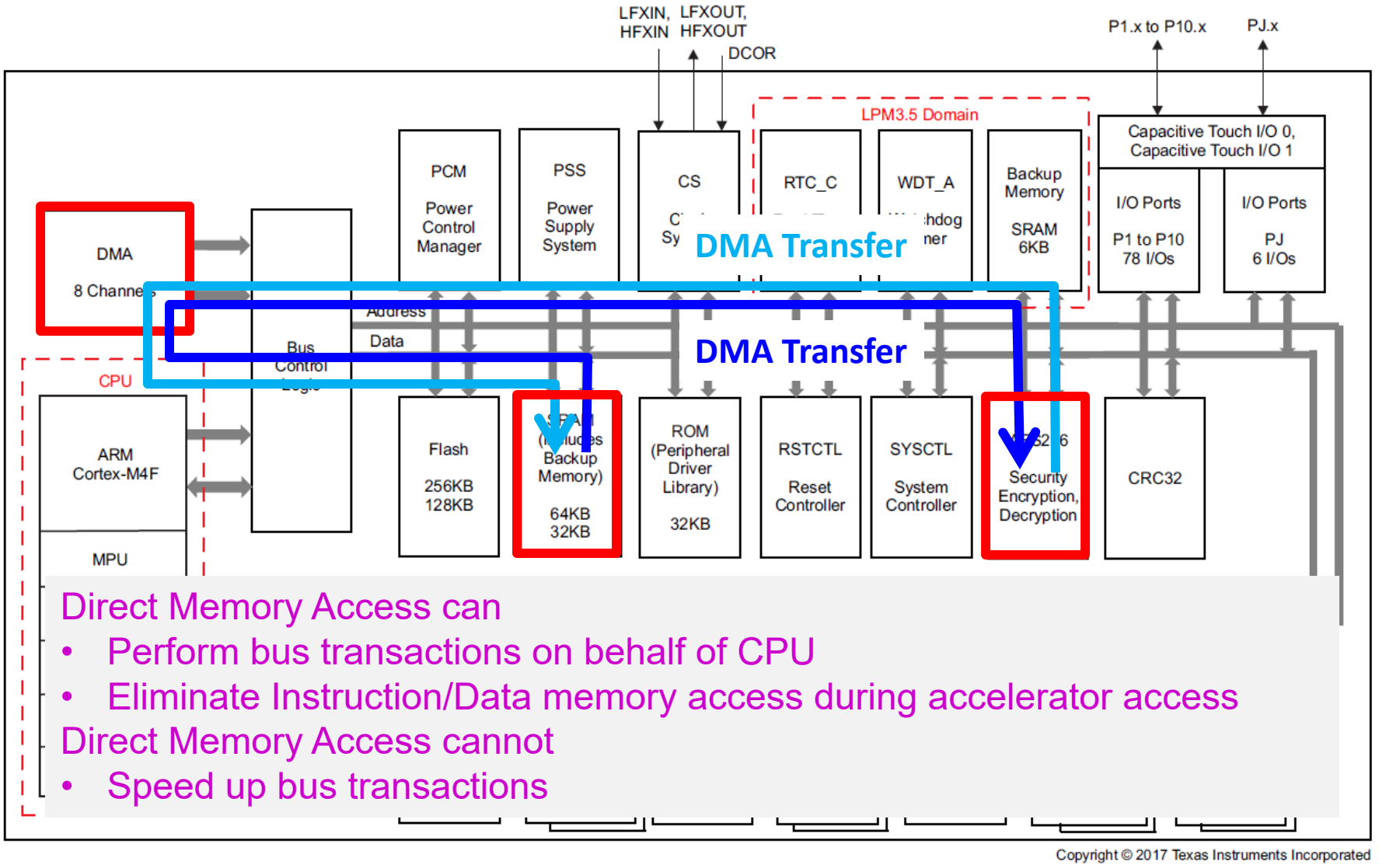
The bus is a bottleneck

- **The bus handles <u>everything</u>**
  - **Data into and from the AES accelerator**
  - **Instruction fetch**
  - **Working variables of the software**

Outline

1. Fundamentals of Parallelism
2. Embedded Architecture of MSP430, MSP432
3. Hardware Acceleration in Embedded Architectures
4. AES Hardware Accelerator
5. Direct Memory Access
6. Power Dissipation
7. Literature

Direct Memory Access



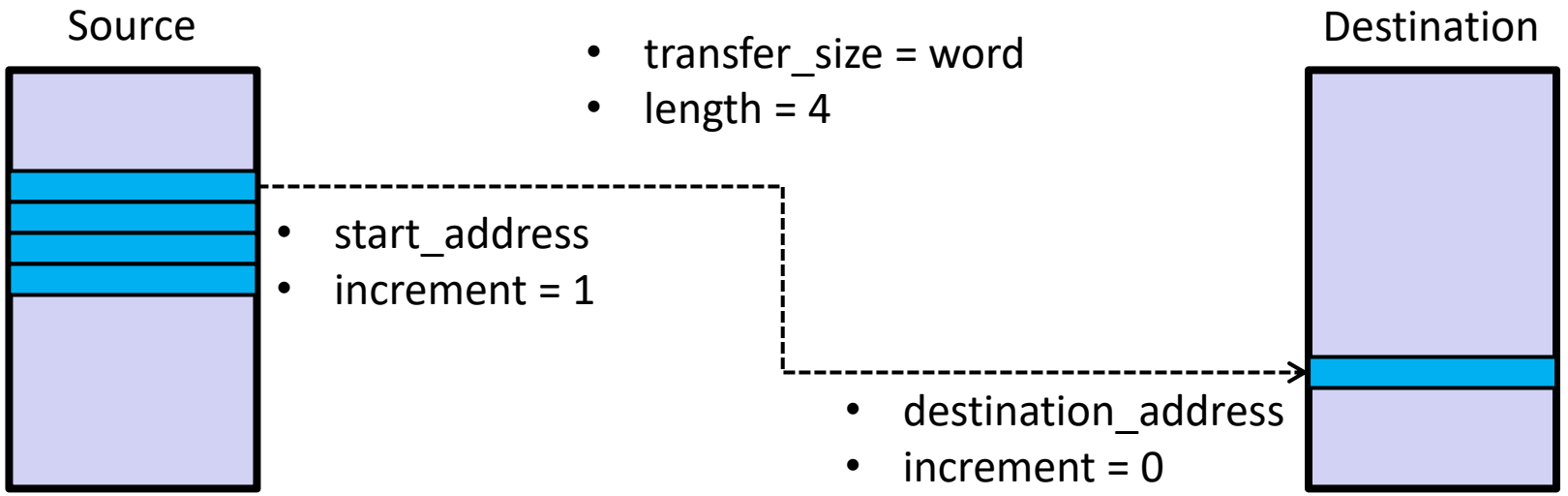
Direct Memory Access can

- Perform bus transactions on behalf of CPU
- Eliminate Instruction/Data memory access during accelerator access

Direct Memory Access cannot

- Speed up bus transactions

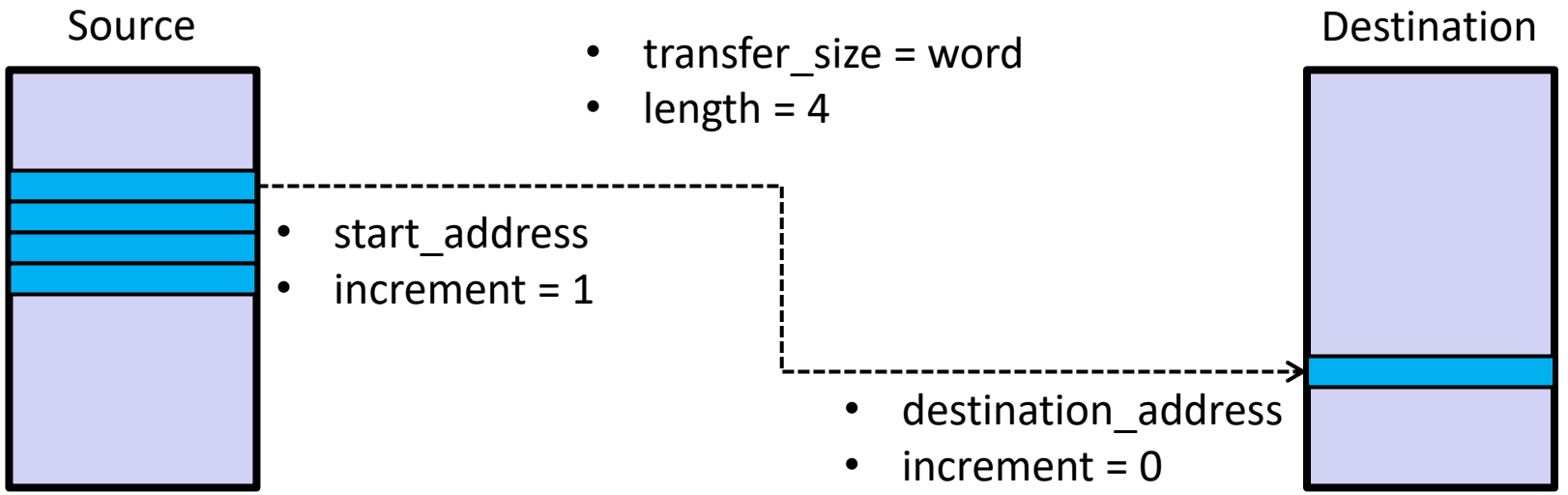
DMA Transfer Parameters



Equivalent Functional Behavior:

```
for (i = 0; i< 4; i++) {
    V = read_word(Start_Address + i);
    write_word(Destination_Address, V);
}
```

DMA Trigger and Completion

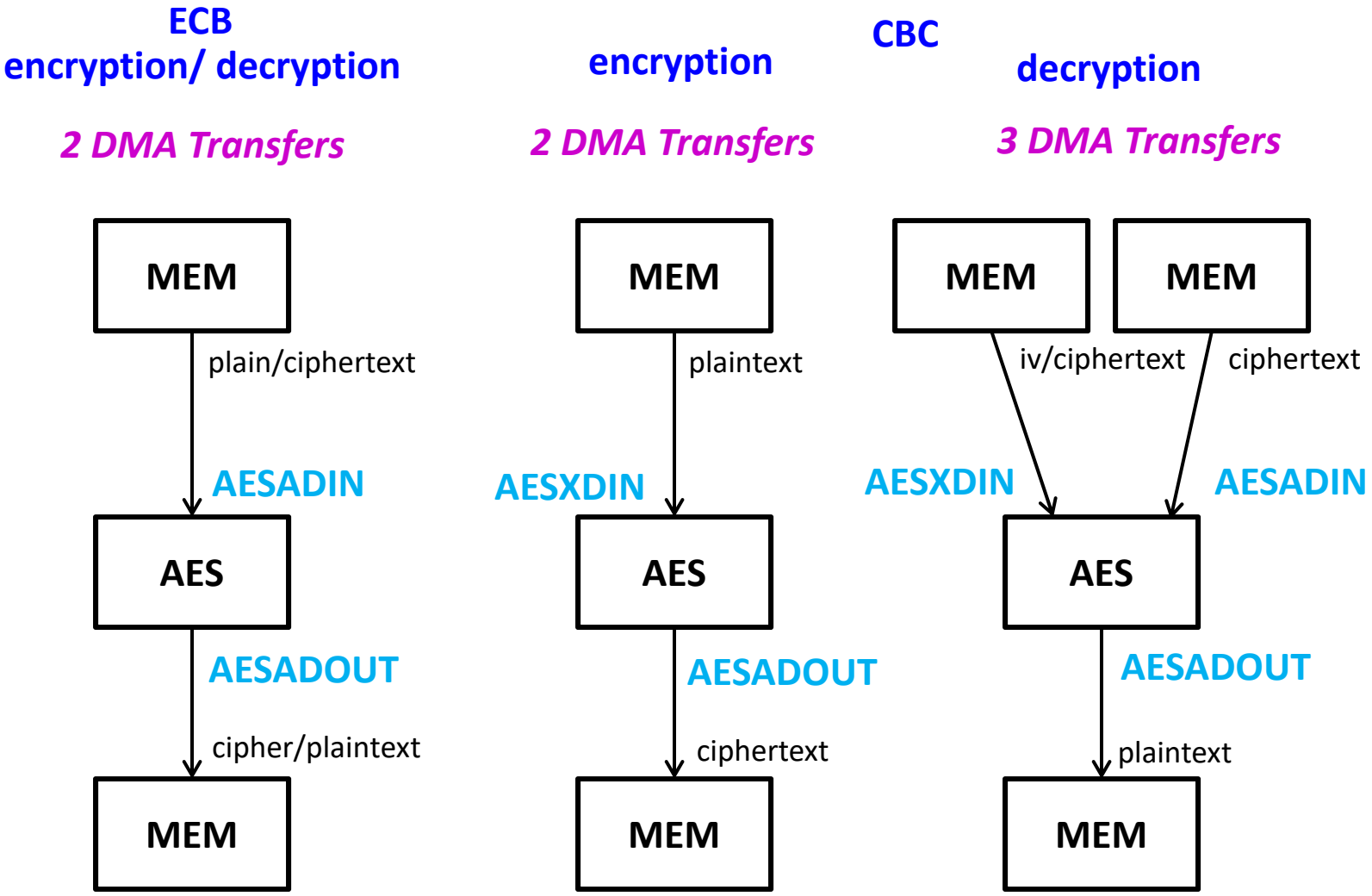


DMA Trigger
(Control Bit or HW)

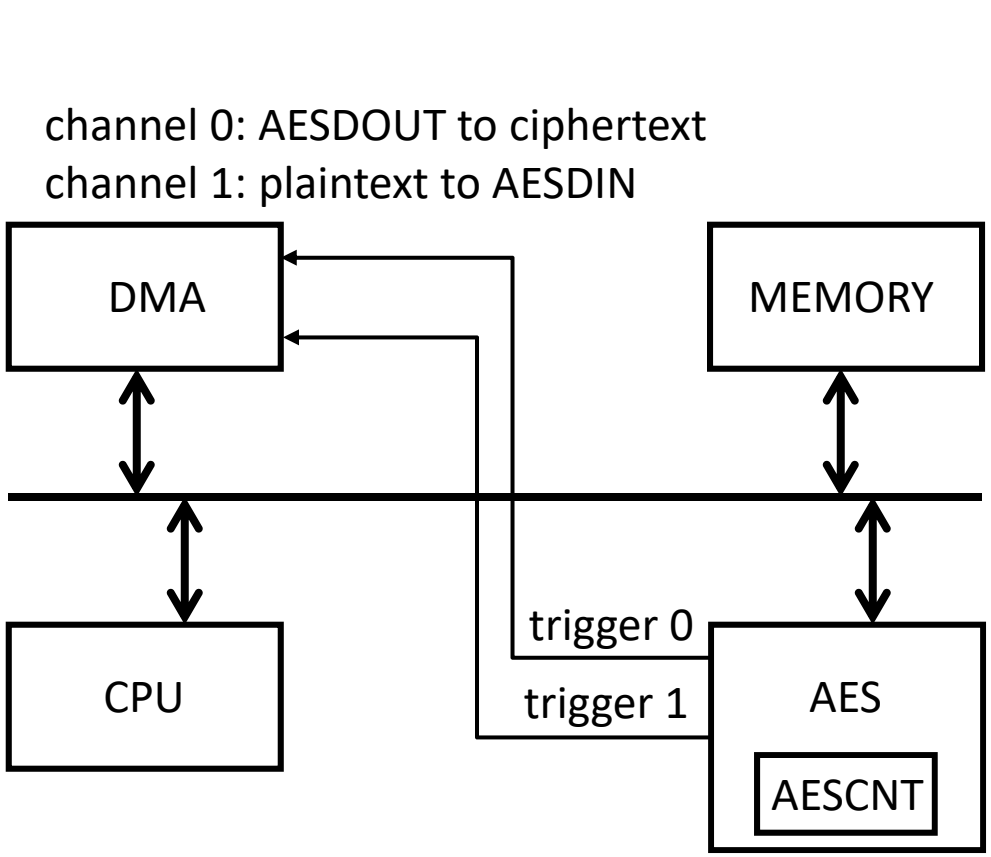
```
Equivalent Functional Behavior:
  for (i = 0; i< 4; i++) {
      wait_for_trigger();
      V = read_word(Start_Address + i);
      write_word(Destination_Address, V);
  }
  assert_completion();
```

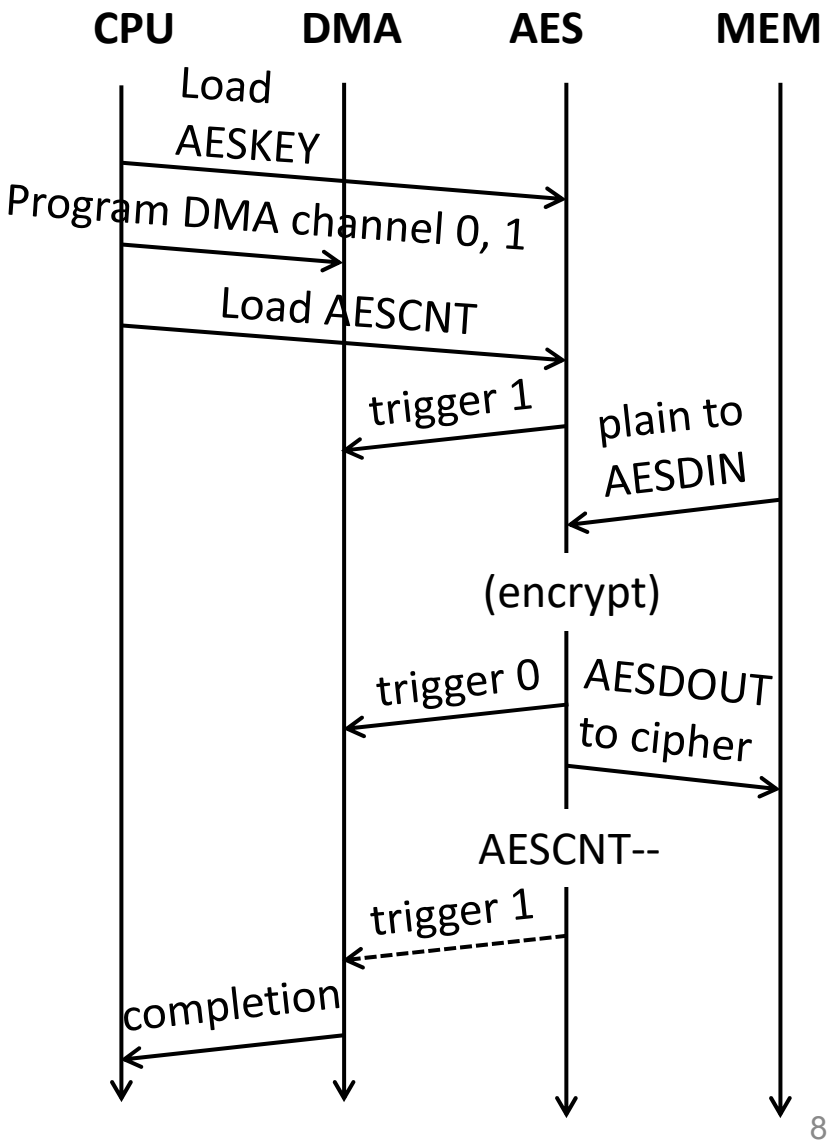
DMA Completion
(Flag or Interrupt)

DMA Transfer on AES Coprocessor

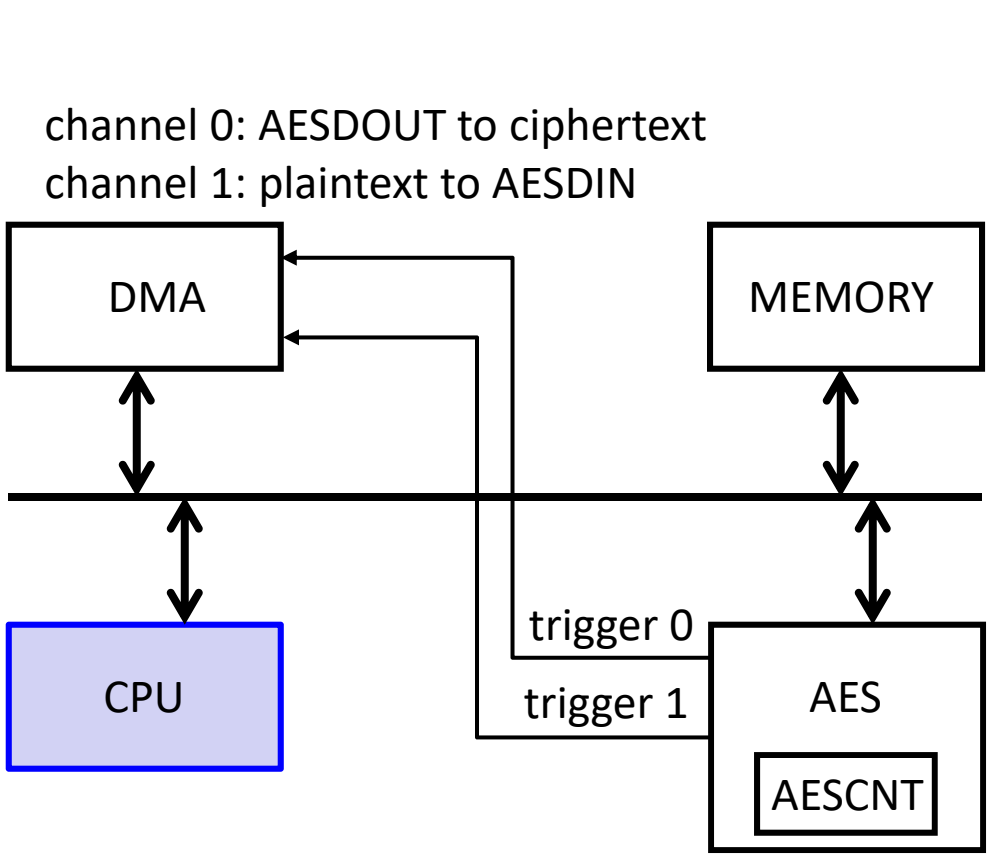


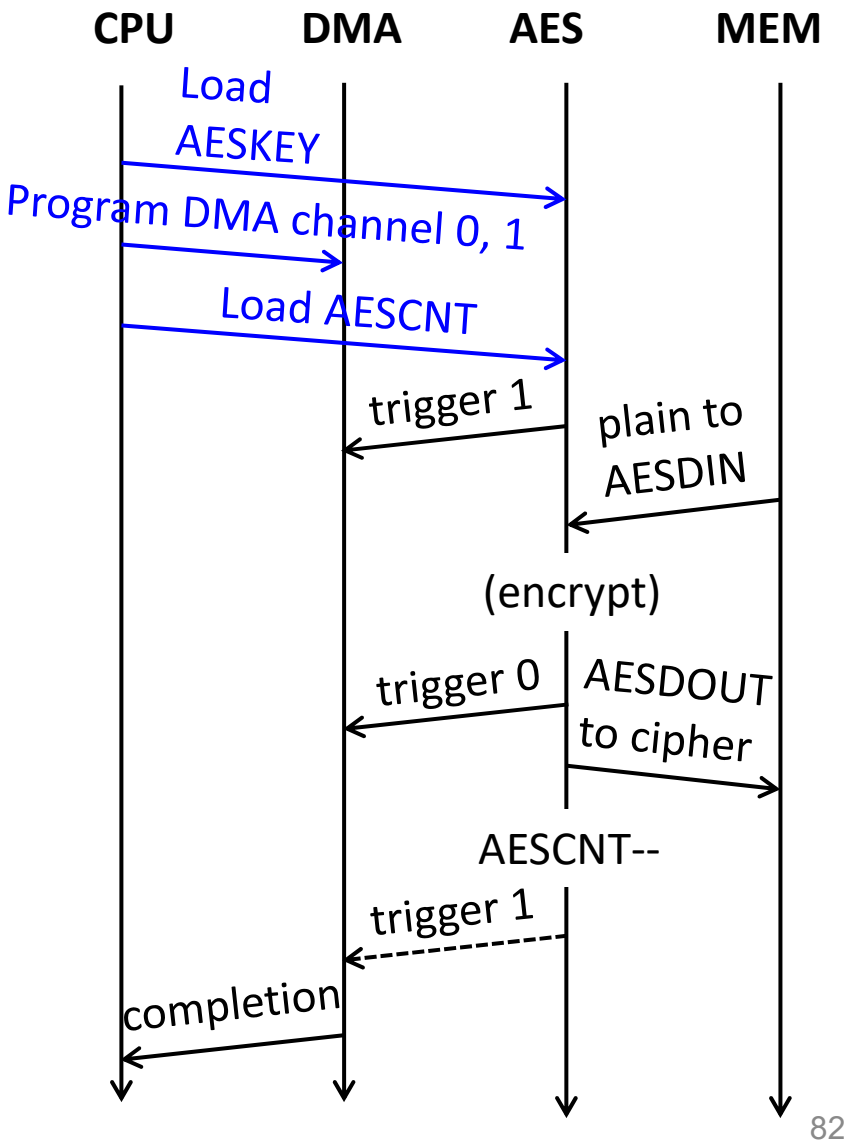
Triggers on AES: ECB



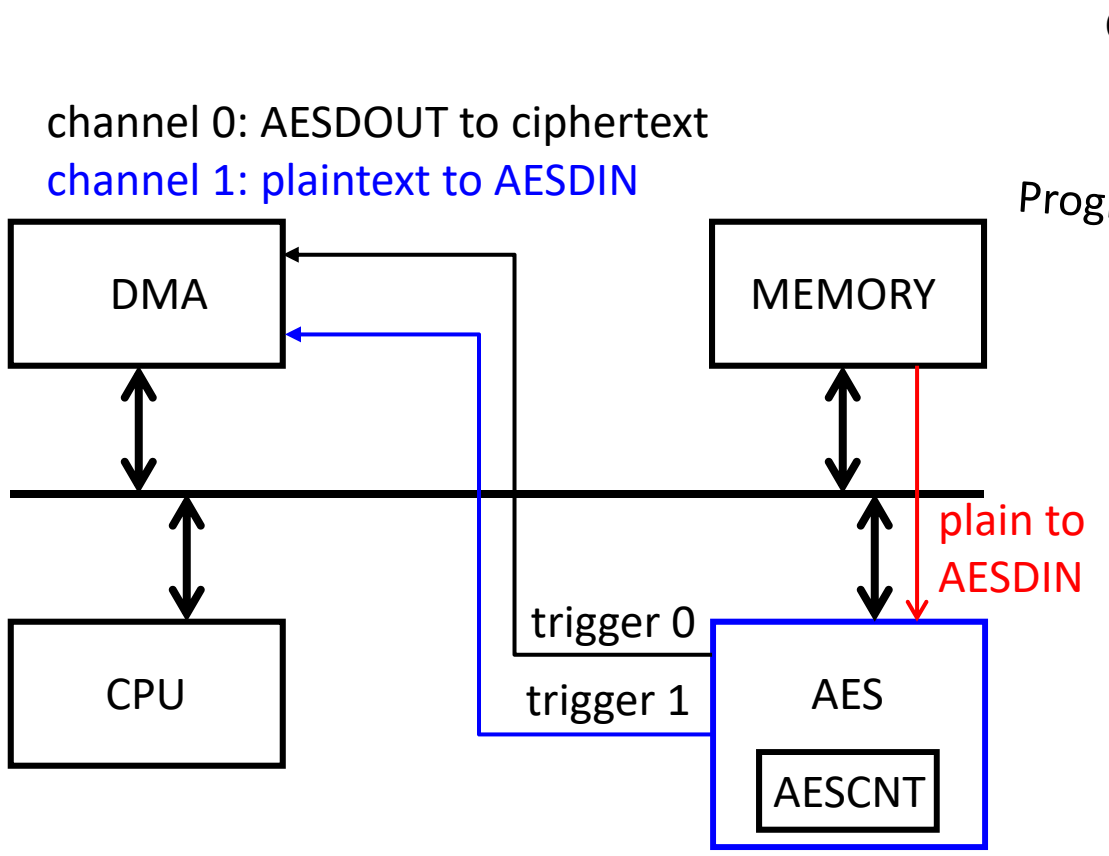


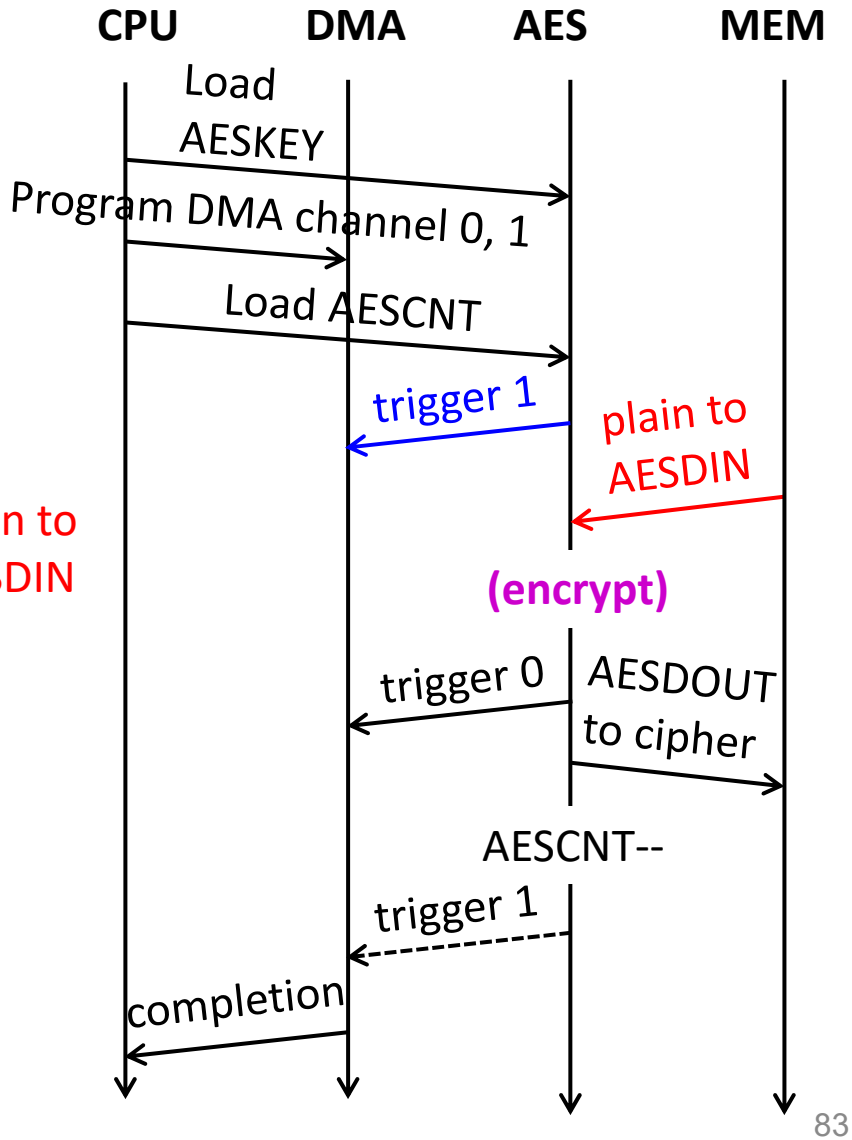
Triggers on AES: ECB



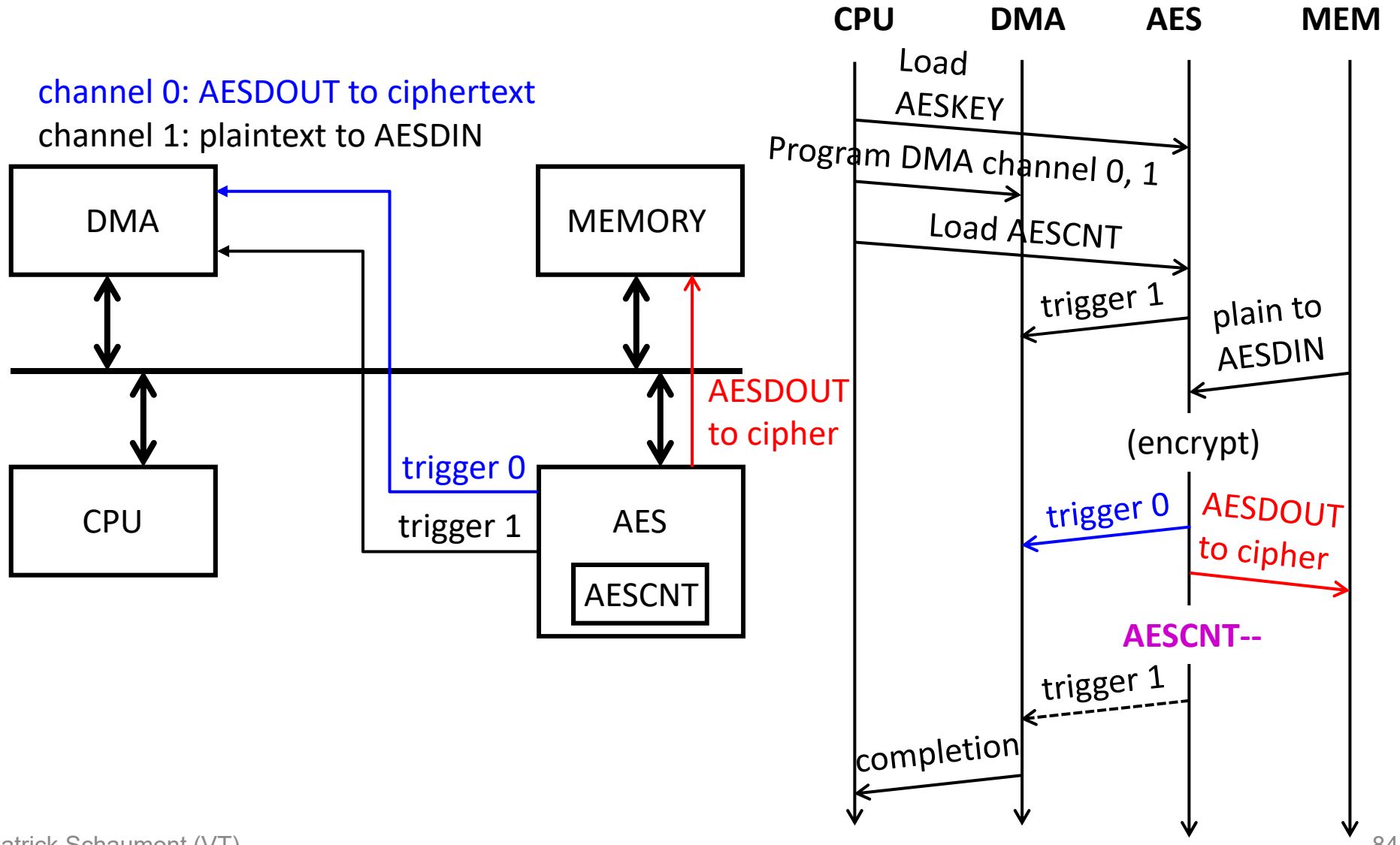


Triggers on AES: ECB





Triggers on AES: ECB



AES ECB using DMA (MSP430)

```
AESACTL0 = AESSWRST;                                          // reset AES
AESACTL0 = (AESACTL0 & ~AESOP) | AESOP_0;                     // set encryption
AESACTL0 = (AESACTL0 & ~AESKL) | AESKL_0;                     // keylength 128
AESACTL0 = (AESACTL0 & ~AESCM) | AESCM__ECB;                  // DMA ECB mode
AESACTL0 = (AESACTL0) | AESCMEN__ENABLE;
for (i=0; i<8; i++) AESAKEY = ((uint16_t *) CipherKey)[i];    // load key
DMACTL0 = DMA0TSEL_11 | DMA1TSEL_12;                          // enable DMA triggers
DMA0CTL = DMADT_0 | DMALEVEL | DMASRCINCR_0 | DMADSTINCR_3; // configure DMA channel 0
__data20_write_long((unsigned long)&DMA0SA, (unsigned long)&AESADOUT);
__data20_write_long((unsigned long)&DMA0DA, (unsigned long)DataAESencrypted);
DMA0SZ   = NUMBLOCKS*8;
DMA0CTL |= DMAEN;
DMA1CTL  = DMADT_0 | DMALEVEL | DMASRCINCR_3 | DMADSTINCR_0; // configure channel 1
__data20_write_long((unsigned long)&DMA1SA, (unsigned long)Data);
__data20_write_long((unsigned long)&DMA1DA, (unsigned long)&AESADIN);
DMA1SZ   = NUMBLOCKS*8;
DMA1CTL |= DMAEN;
AESACTL1 = NUMBLOCKS;                                          // start encryption

while (!(DMA0CTL & DMAIFG)) ;                                  // wait for completion
DMAIV |= 0;

DMA0CTL = DMA0CTL & (~DMAEN);                                  // disable DMA
DMA1CTL = DMA1CTL & (~DMAEN);
```

Performance of AES Accelerator

Design	Accelerator MSP430	Accelerator MSP432
AES Enc ECB	250	450
AES Dec ECB	250	548
AES Enc CBC	282	452
AES Dec CBC	423	622

Design	Accel+DMA MSP430	Accel+DMA MSP432
AES Enc ECB	**203**	***490***
AES Dec ECB	**203**	***584***
AES Enc CBC	**203**	***491***
AES Dec CBC	**203**	***558***

Max optimization level (speed)
Per-block cycles counts excl DMA programming overhead
benchmarking 1K-block (MSP 432) or 32 block (MSP430)

Performance of AES Accelerator

Close to lower bound:
2 cycles per bus transfer (MSP430)
8+8 transfers takes 32 cycles
AES core needs 168 cycles
168 + 32 = 200

Requires further inspection. Possible bus contention w CPU?

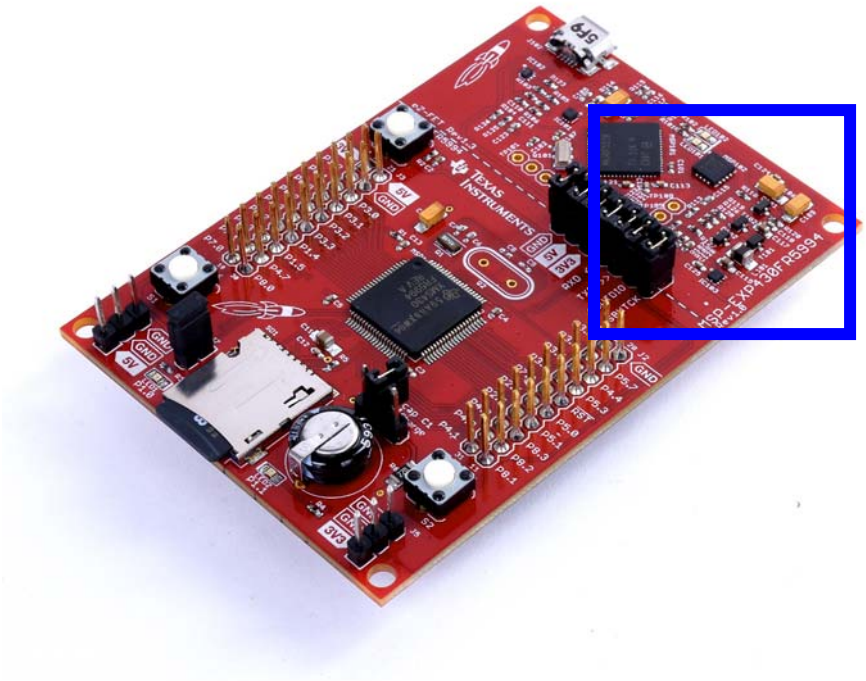
Design	Accel+DMA MSP430	Accel+DMA MSP432
AES Enc ECB	**203**	***490***
AES Dec ECB	**203**	***584***
AES Enc CBC	**203**	***491***
AES Dec CBC	**203**	***558***

Max optimization level (speed)
Per-block cycles counts excl DMA programming overhead
benchmarking 1K-block (MSP 432) or 32 block (MSP430)

Outline

1. Fundamentals of Parallelism
2. Embedded Architecture of MSP430, MSP432
3. Hardware Acceleration in Embedded Architectures
4. AES Hardware Accelerator
5. Direct Memory Access
6. Power Dissipation
7. Literature

EnergyTrace



US 20130154594A1

(19) **United States**

(12) **Patent Application Publication**

Zipperer et al.

(10) **Pub. No.: US 2013/0154594 A1**

(43) **Pub. Date: Jun. 20, 2013**

(54) **ELECTRONIC DEVICE AND METHOD FOR POWER MEASUREMENT**

(75) Inventors: **Johann Zipperer**, Unterschleissheim (DE); **Peter Weber**, Allershausen (DE)

(73) Assignee: **Texas Instruments Incorporated**, Dallas, TX (US)

(21) Appl. No.: **13/329,073**

(22) Filed: **Dec. 16, 2011**

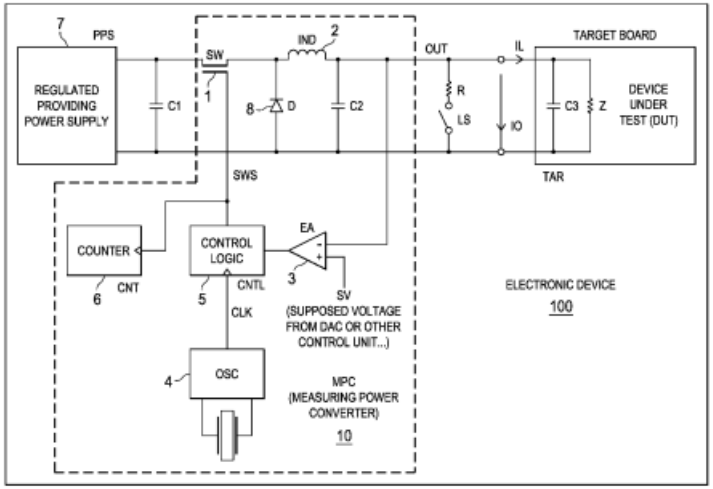
Publication Classification

(51) **Int. Cl.**
G05F 1/10 (2006.01)

(52) **U.S. Cl.**
USPC **323/282**

(57) **ABSTRACT**

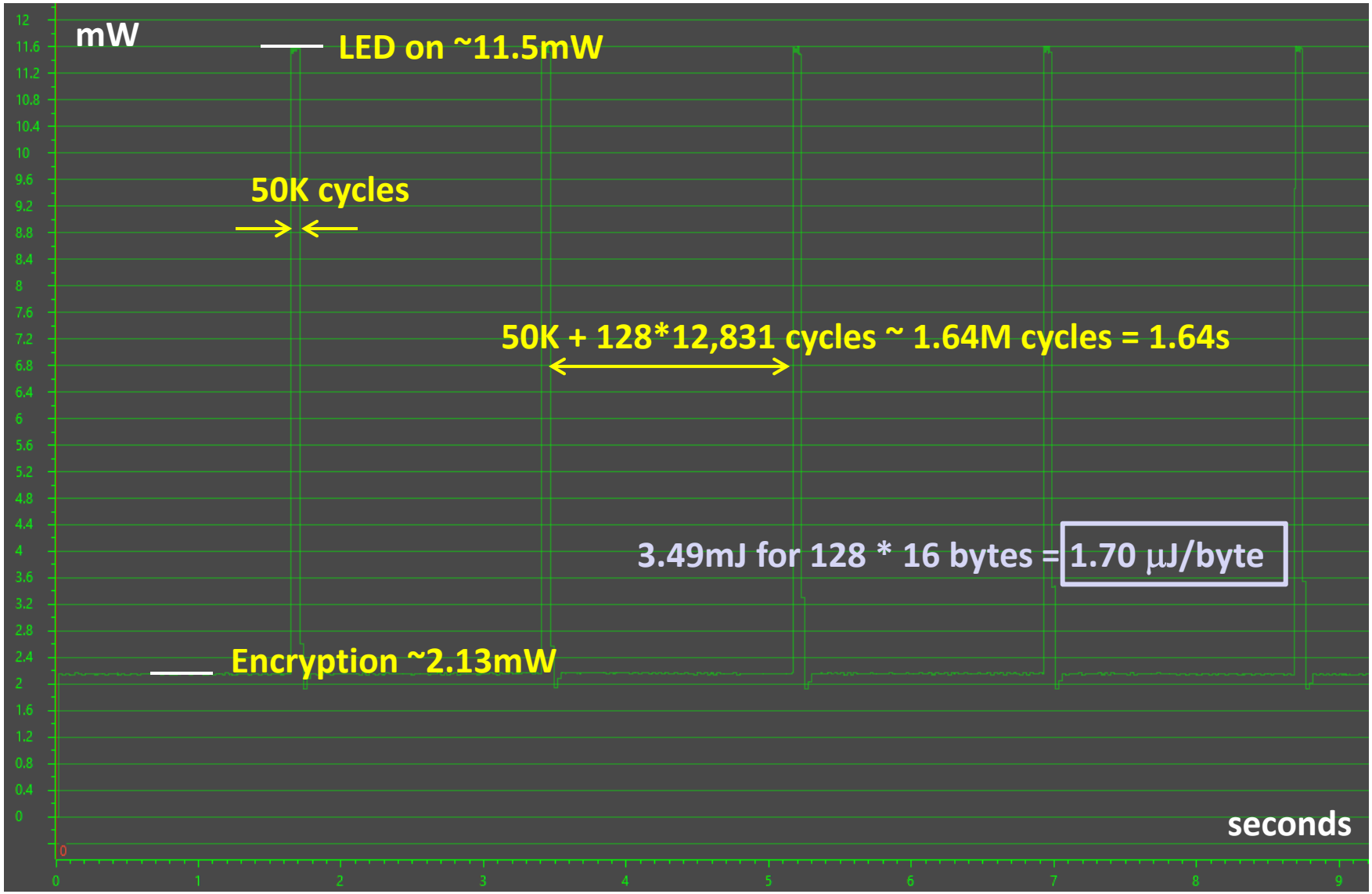
The invention relates to an electronic device comprising a switched mode power converter comprising a switched transistor, an inductor and an error amplifier. The switched transistor is configured to switch a current through the inductor. The error amplifier is configured to control the switching of the switched transistor to convert a primary voltage applied at the input into a secondary voltage at the output of the switched mode power converter. The electronic device further comprises an oscillator, a control logic stage and a digital counter. The control logic stage is coupled to receive a clock signal from the oscillator and to generate switching signals for the switched transistor in form of ON-time pulses with a constant ON-time according to a pulse density scheme. The counter is configured to count the number of ON-time pulses for determining the consumed power based on the number of ON-time pulses per time.



Measurement Loop MSP430 Ref

```
P1OUT &= ~BIT0;
P1DIR |= BIT0;                // GPIO LED
PM5CTL0 &= ~LOCKLPM5;
P1OUT = 0;                    // clear LED

uint32_t k;
while (1) {
   for (k=0; k<128; k++) {
      AES_ECB_encrypt(&SWAES, Data);
   }

   P1OUT ^= BIT0;              // toggle LED
   __delay_cycles(50000);
   P1OUT ^= BIT0;

   __delay_cycles(50000);
}
```

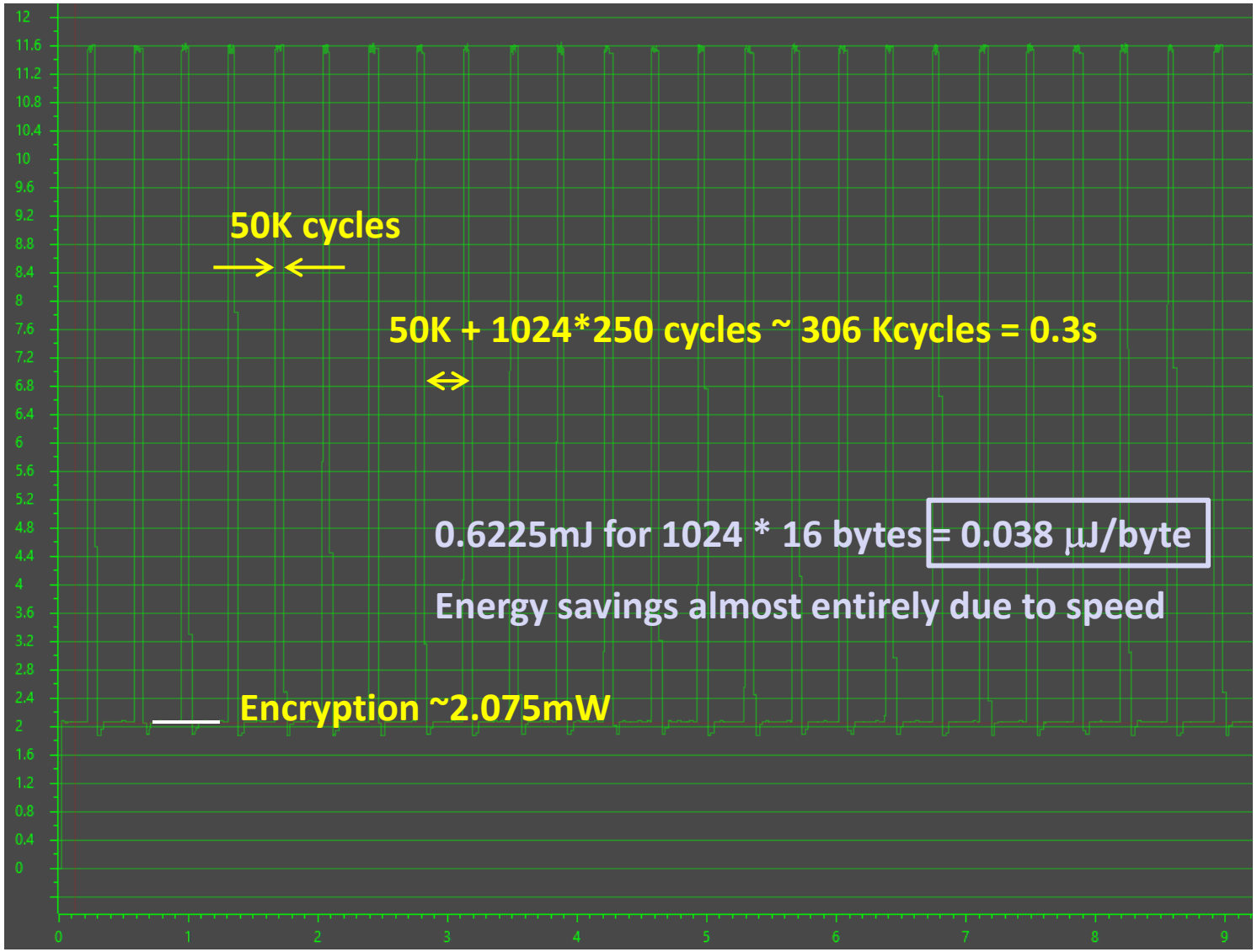
Measurement Loop MSP430 Ref



Measurement Loop MSP430 AES

```
uint32_t k;
while (1) {
  for (k=0; k<1024; k++) {
    for (i=0; i<8; i++)
      AESADIN = ((uint16_t *) Data)[i];
    while (AESASTAT & AESBUSY) ;
    for (i=0; i<8; i++)
      ((uint16_t *) DataAESencrypted)[i] = AESADOUT;
  }

  P1OUT ^= BIT0;                    // toggle LED
  __delay_cycles(50000);
  P1OUT ^= BIT0;

  __delay_cycles(50000);
  }
}
```

Measurement Loop MSP430 AES



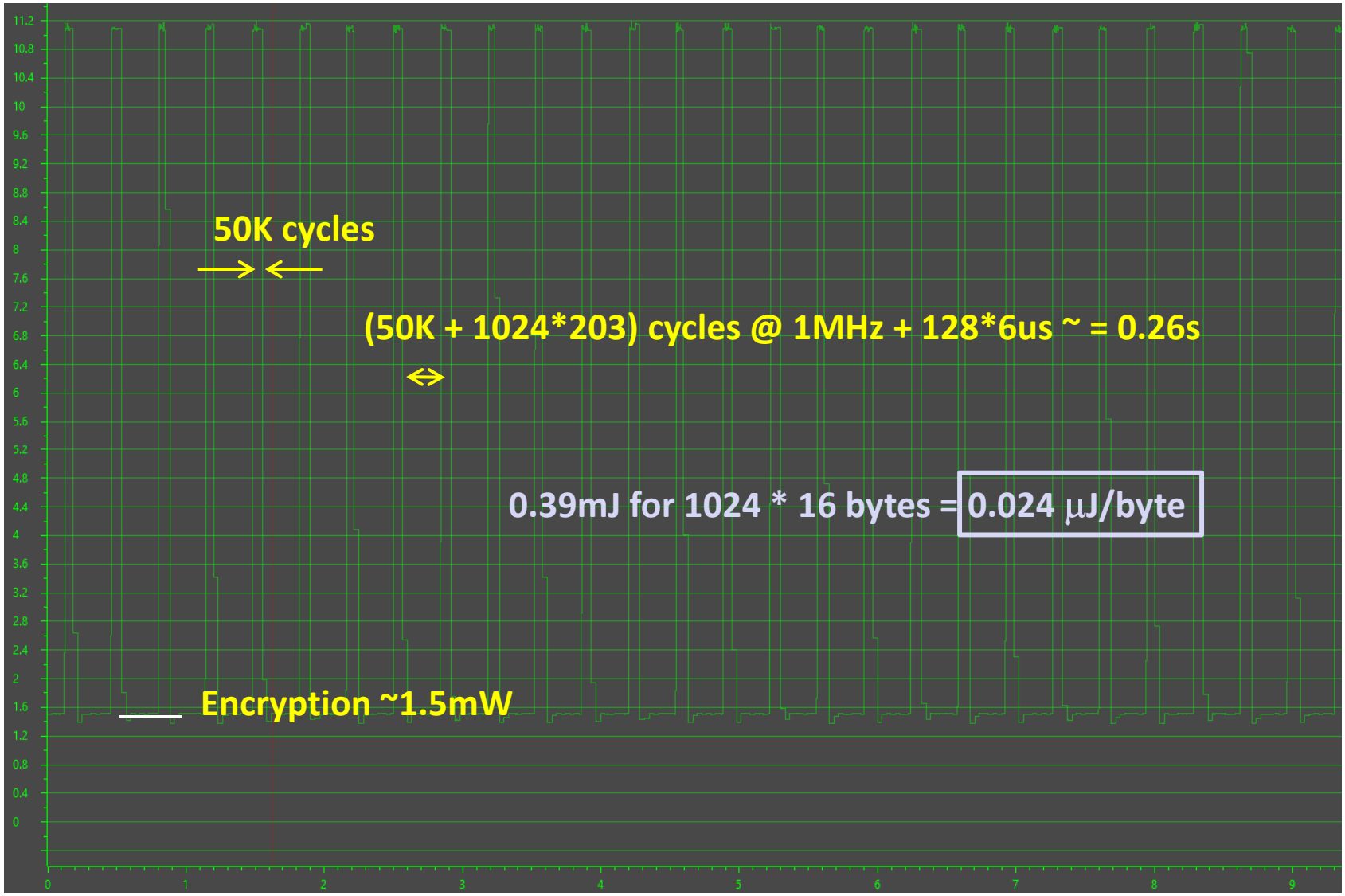
Measurement Loop MSP430 AES DMA

```
uint16_t k;
while (1) {
  for (k=0; k<1024/NUMBLOCKS; k++) {
    // DMA Channel 0 programming
    // ...
    DMA0CTL &= ~DMAIFG;
    DMA0CTL |= DMAIE;   // enable completion interrupts
    DMA0CTL |= DMAEN;

    // DMA Channel 1 programming
    // ...
    AESACTL1 = NUMBLOCKS;

    __bis_SR_register(LPM1_bits + GIE); // low-power, turn on interrupts
  }

  P1OUT ^= BIT0;                        // toggle LED
  __delay_cycles(50000);
  P1OUT ^= BIT0;

  __delay_cycles(50000);
}
```

MSP430 AES DMA Completion Interrupt

```
#pragma vector=DMA_VECTOR
__interrupt void DMA_ISR(void) {
  switch(__even_in_range(DMAIV,16))
  {
    case 0: break;
    case 2: // DMA Channel 0 completion interrupt
            __bic_SR_register_on_exit(LPM1_bits ); // Disable LPM mode
            break;
    default: break;
  }
}
```

Measurement Loop MSP430 AES DMA



Speed/Power/Energy

	Time	Power	Energy
Mode	**Cycles**	**mW @1MHz**	**uJ/byte**
AES Enc ECB (SW)	12,831	2.13	1.7
AES Enc ECB (Acc)	250	2.075	0.038
AES Enc ECB (Acc+DMA)	203	1.5	0.024

Mode	Speedup	Relative	Relative
AES Enc ECB (SW)	1	1	1
AES Enc ECB (Acc)	51	0.97	0.022
AES Enc ECB (Acc+DMA)	63	0.70	0.014

Outline

1. **Fundamentals of Parallelism**
2. **Embedded Architecture of MSP430, MSP432**
3. **Hardware Acceleration in Embedded Architectures**
4. **AES Hardware Accelerator**
5. **Direct Memory Access**
6. **Power Dissipation**
7. **Literature**

Literature and References

- **MSP430FR5994 Launchpad**

  http://www.ti.com/tool/MSP-EXP430FR5994

- **MSP432P401R Launchpad**

  http://www.ti.com/tool/MSP-EXP432P401R

- **Code Composer Studio**

  http://www.ti.com/tool/CCSTUDIO

Thank You!

Questions?

Patrick Schaumont

schaum@vt.edu