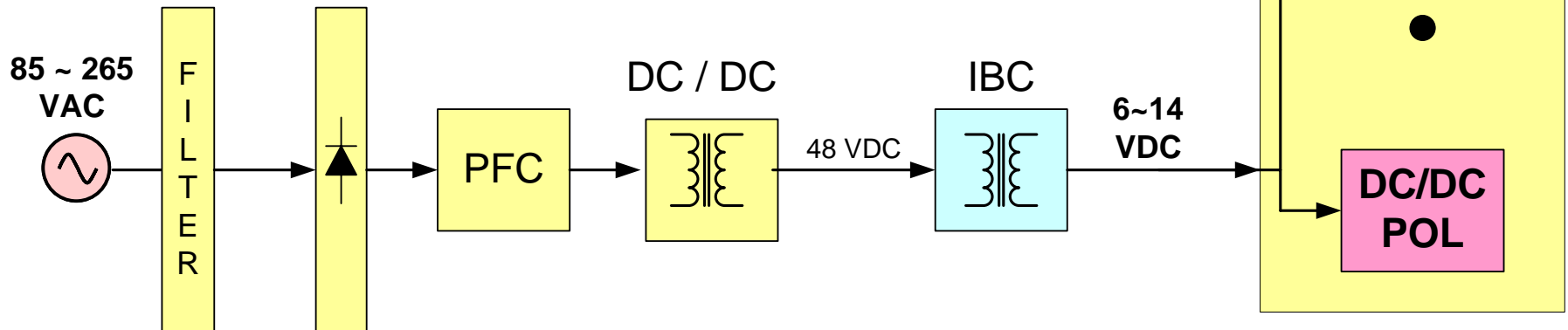

Multi-phase & Multi-output DC/DC power supplies using Software Programmable Digital Controllers

David Figoli
Systems and Device Architect
Texas Instruments – Houston

d-figoli@ti.com

Multi-Rail DC/DC conversion

- ❑ Telecom infrastructure
- ❑ Base stations
- ❑ Servers
- ❑ Routers
- ❑ Workstations



Requirements broader & more challenging

Wide range in Process tech (350 nm ~ 90 nm)

→ More voltage outputs (1.0, 1.2, 1.8, 3.3, 5V....)

High current, low ripple

→ More phases i.e. multi-phase interleaved

Both needed on same system boards

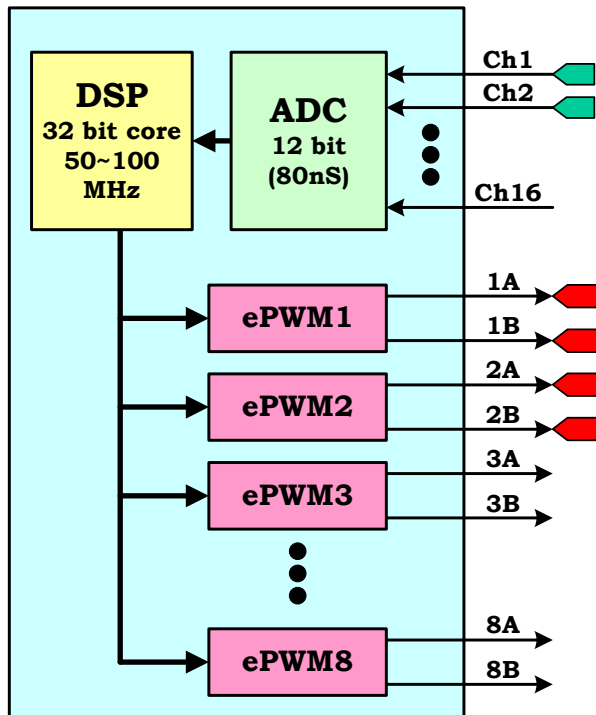
- Soft-start / shut-down
- Sequencing
- Margining (+/- % adjust via host)
- Fault management / reporting
- Fault recovery (e.g. on-fly N to N-1 phase shedding)
- Fault prediction – minimize down time

System considerations & Design flow

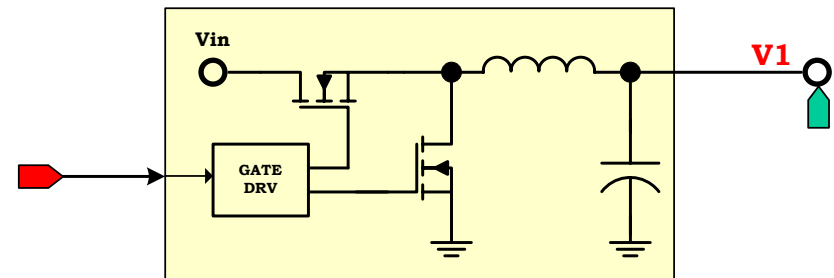
Solution Mapping

- ❑ Resource Allocation
- ❑ PWM waveform generation
- ❑ ADC sampling bandwidth
- ❑ CPU bandwidth for Control processing
- ❑ CPU bandwidth for System management

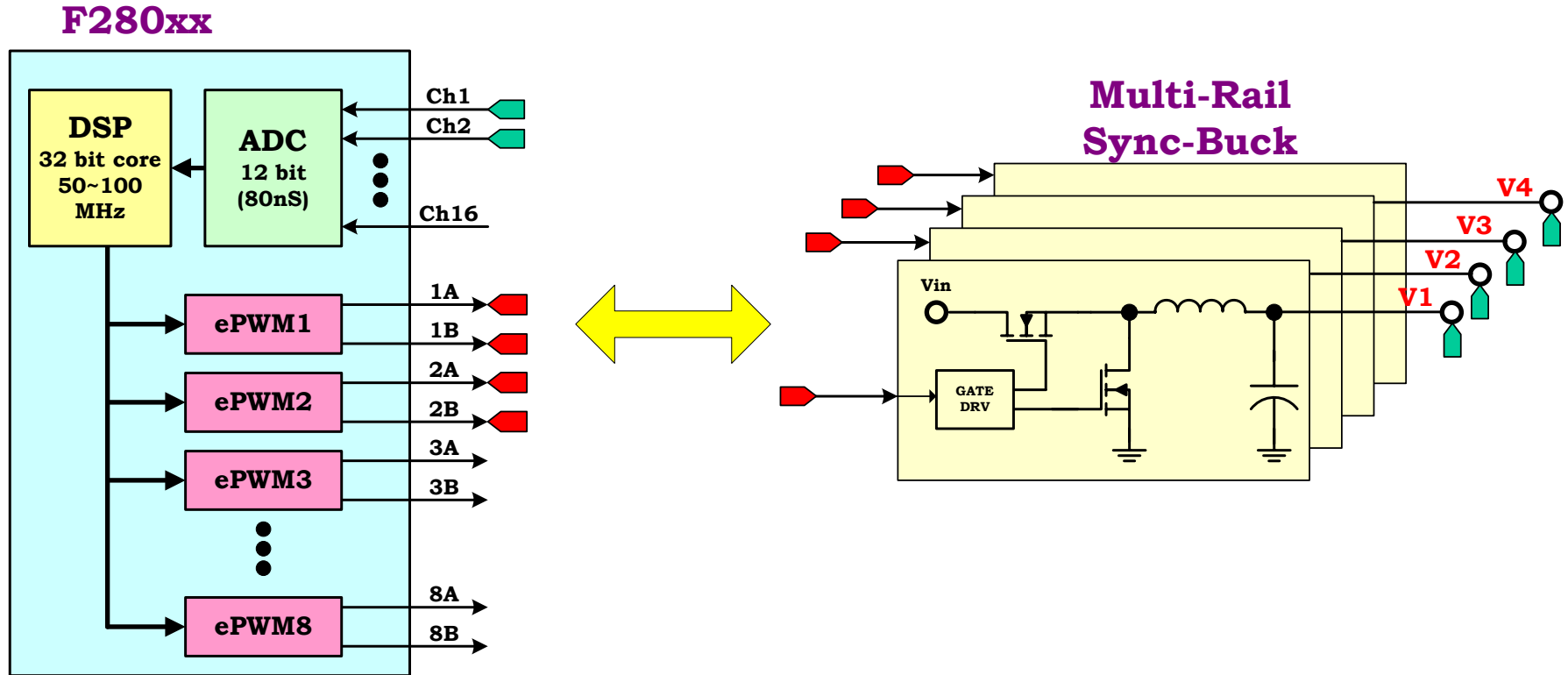
F280xx



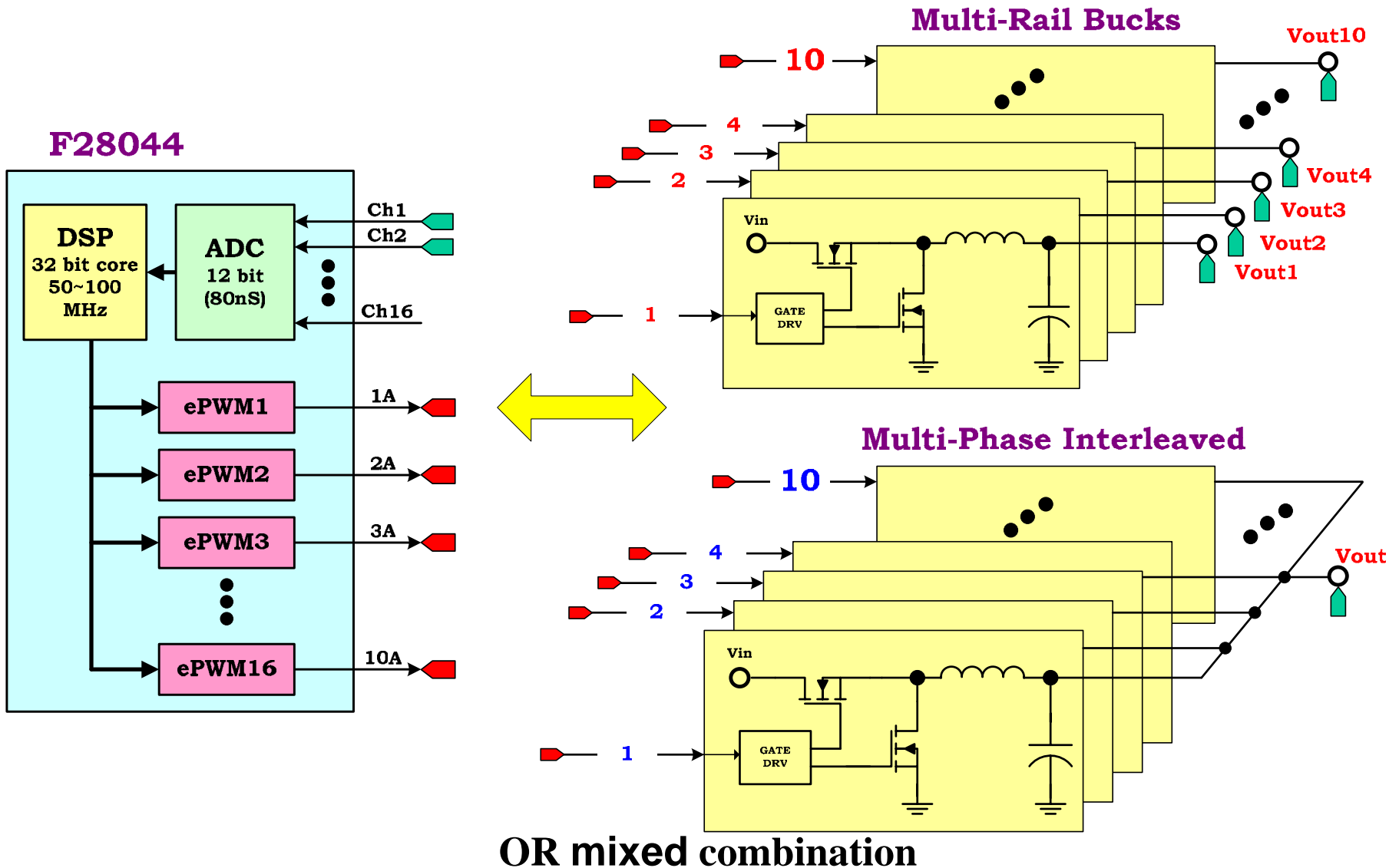
Sync-Buck



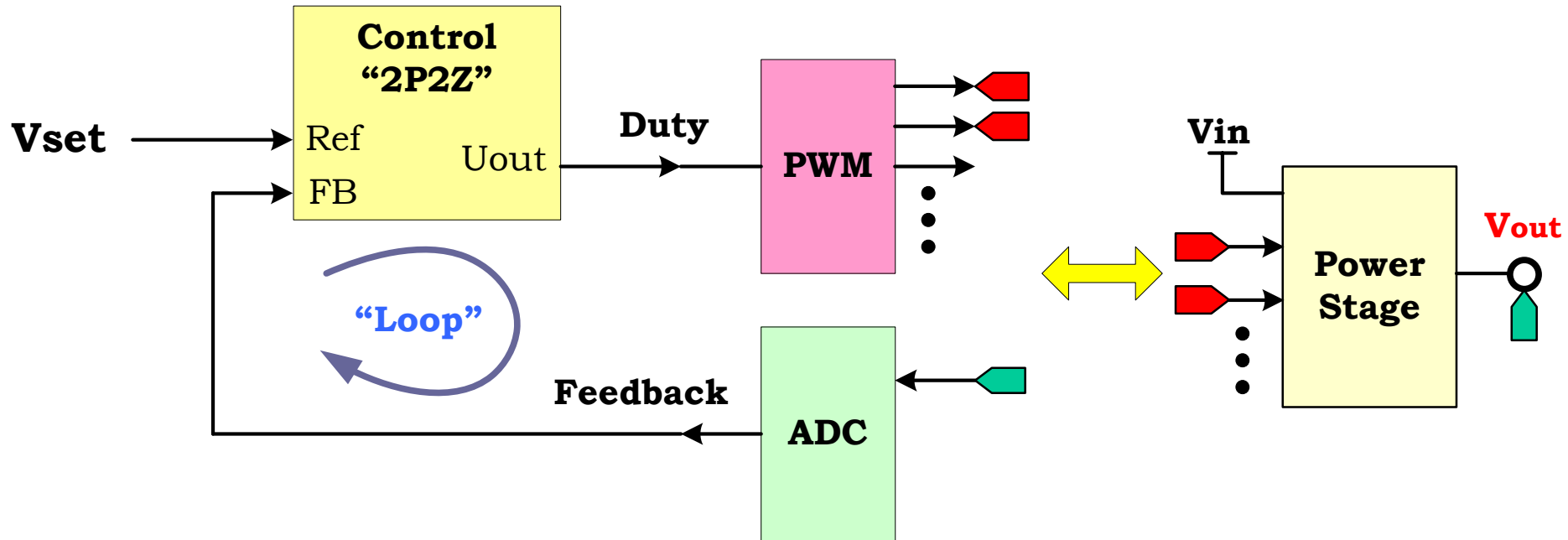
System Mapping (cont.)



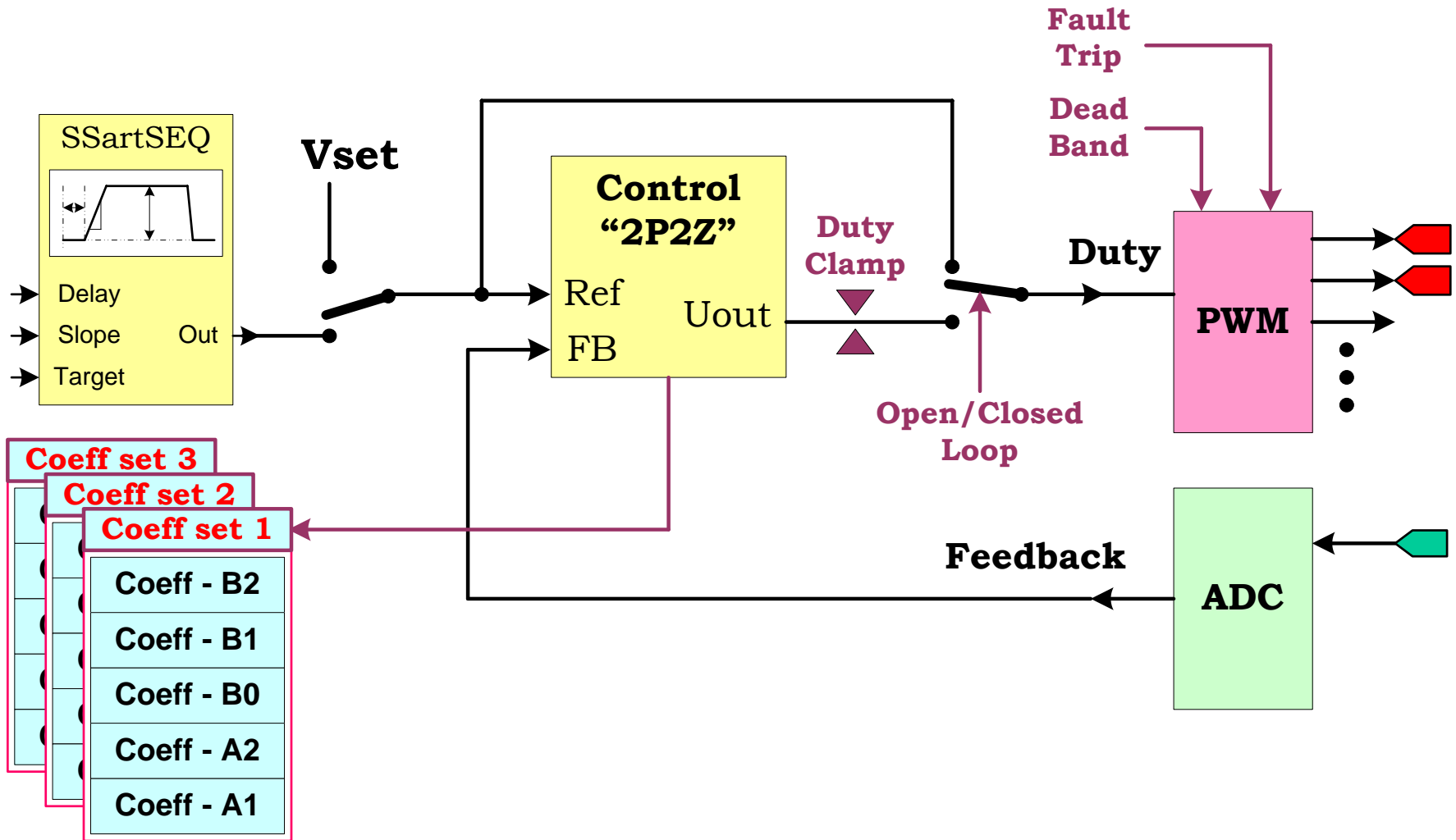
System Mapping (cont.)



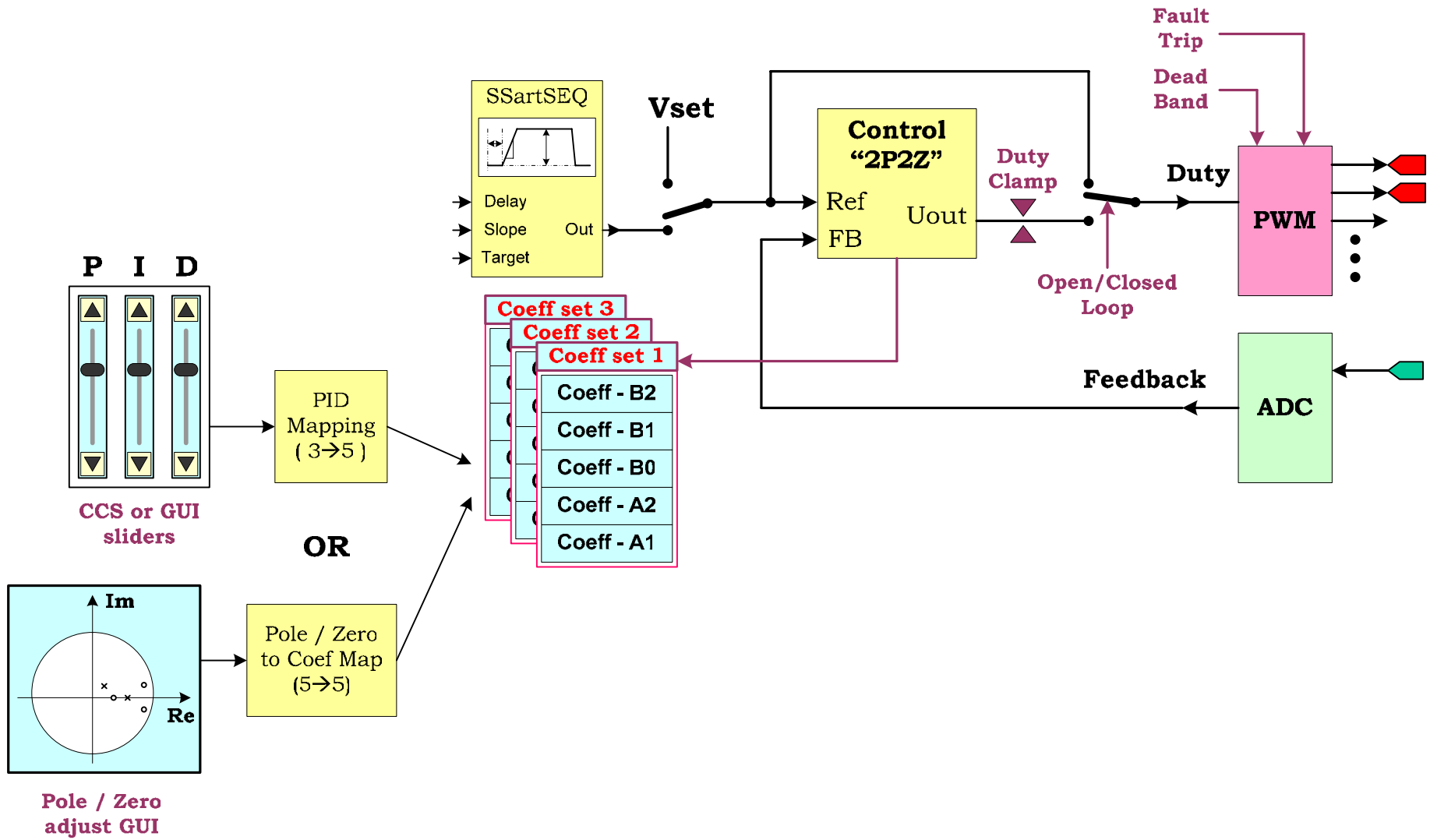
The “Closed Loop”



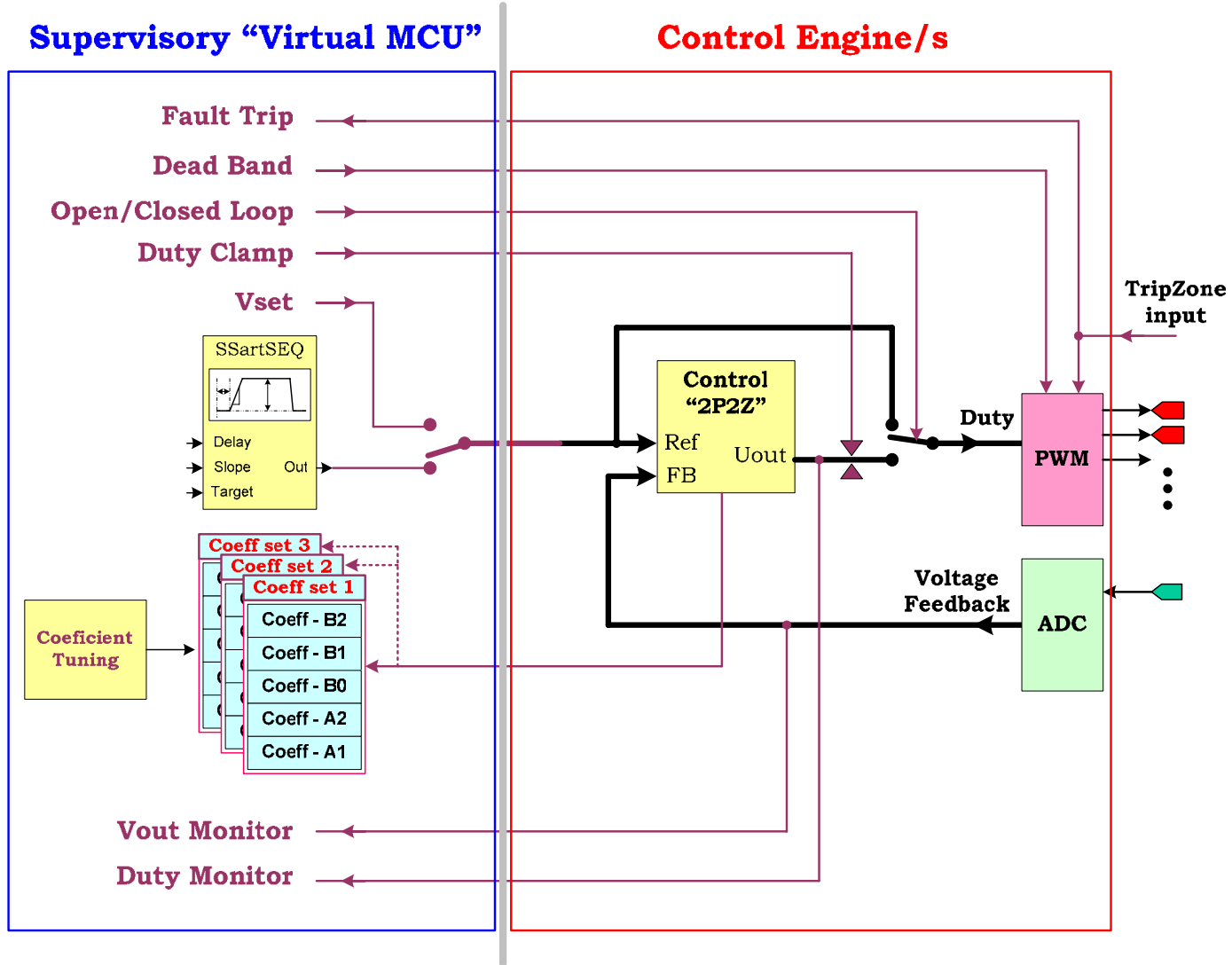
Managing the “Closed Loop”



Tuning the “Closed Loop”



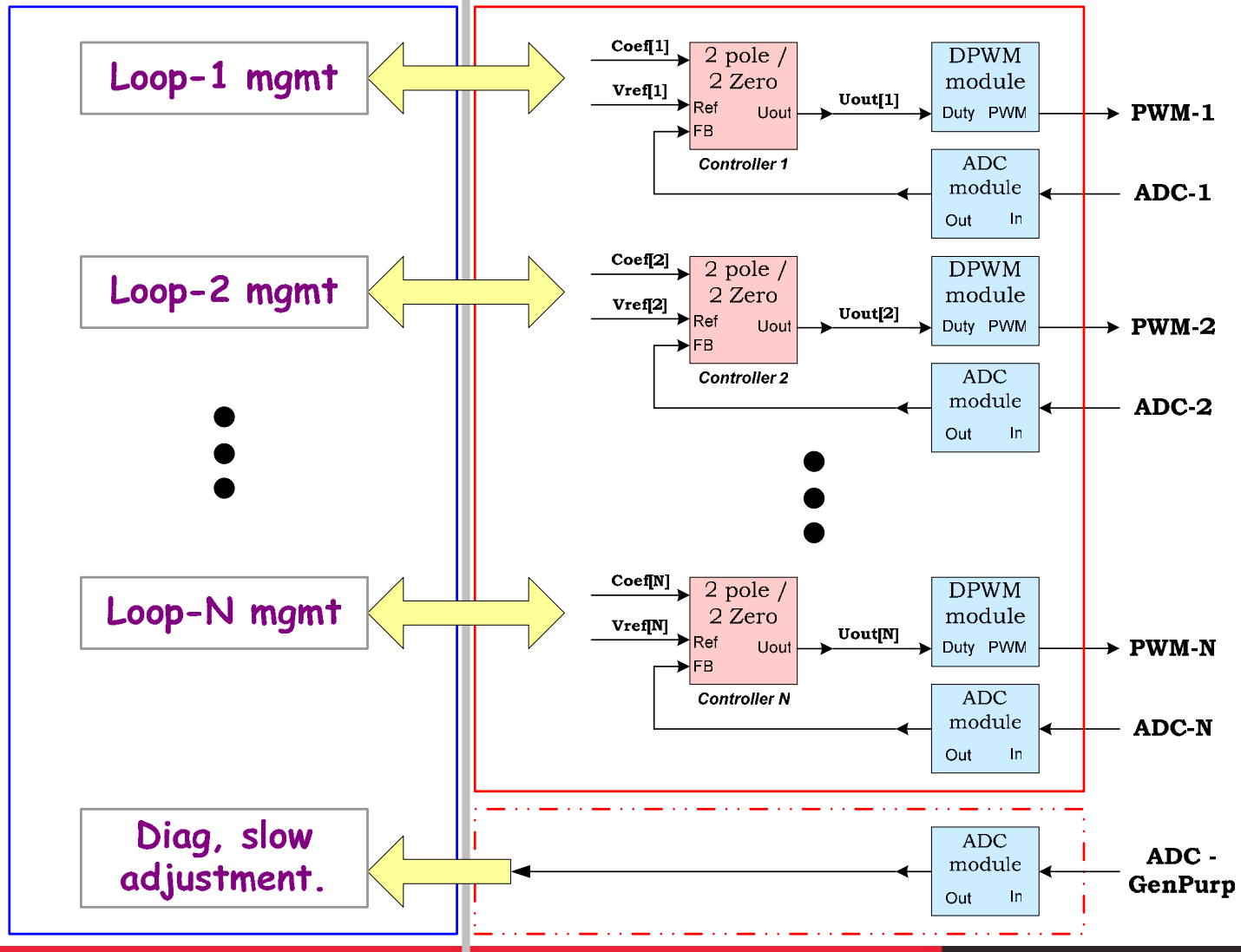
MCU “virtualization”



Multi-loop scalable

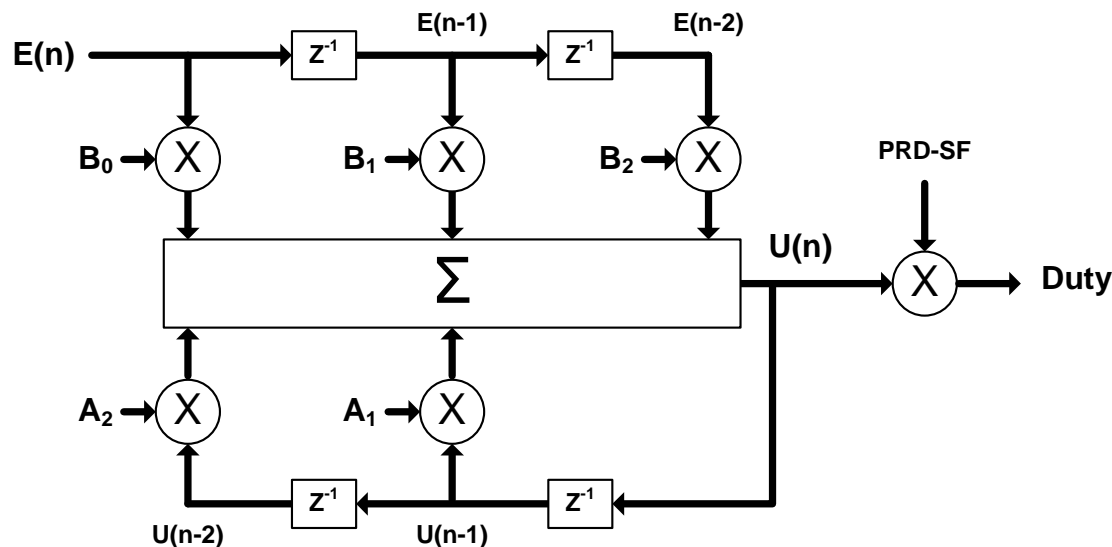
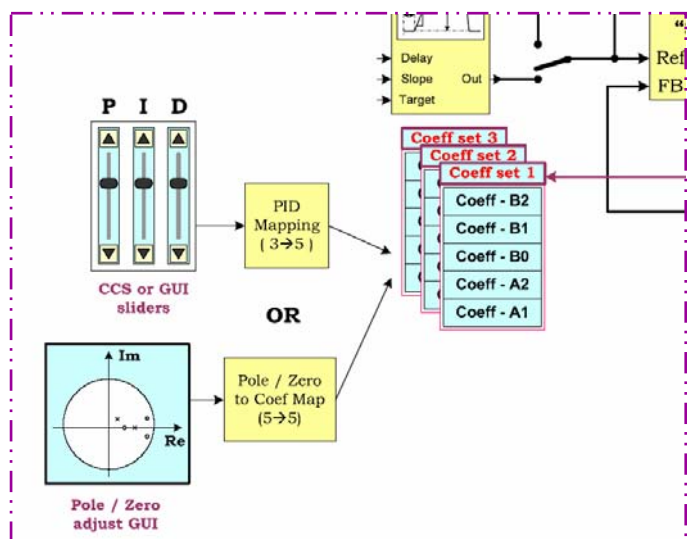
Supervisory "Virtual MCU"

Control Engine/s



Interactive tuning – good first step

$$U(n) = B_0 * E(n) + B_1 * E(n-1) + B_2 * E(n-2) + A_1 * U(n-1) + A_2 * U(n-2)$$



Efficient CPU utilization

PID – intuitive / interactive

We can also write the controller in transfer function form:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} = \frac{b_0 z^2 + b_1 z + b_2}{z^2 - z}$$

Compare with the General 2P2Z transfer function:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2}$$

We can see that PID is nothing but a special case of 2P2Z control where:

$$a_1 = -1 \quad \text{and} \quad a_2 = 0$$

Change PID coeff. “on fly” in back-ground loop

```
// Coefficient init
Coef2P2Z_1[0] = Dgain * 67108; // B2
Coef2P2Z_1[1] = (Igain - Pgain - Dgain - Dgain)*67108; // B1
Coef2P2Z_1[2] = (Pgain + Igain + Dgain)*67108; // B0
Coef2P2Z_1[3] = 0; // A2
Coef2P2Z_1[4] = 67108864; // A1
Coef2P2Z_1[5] = Dmax[1] * 67108; // Clamp Hi limit (Q26)
Coef2P2Z_1[6] = 0x00000000; // Clamp Lo
```

CPU MIPS Allocation & Partitioning

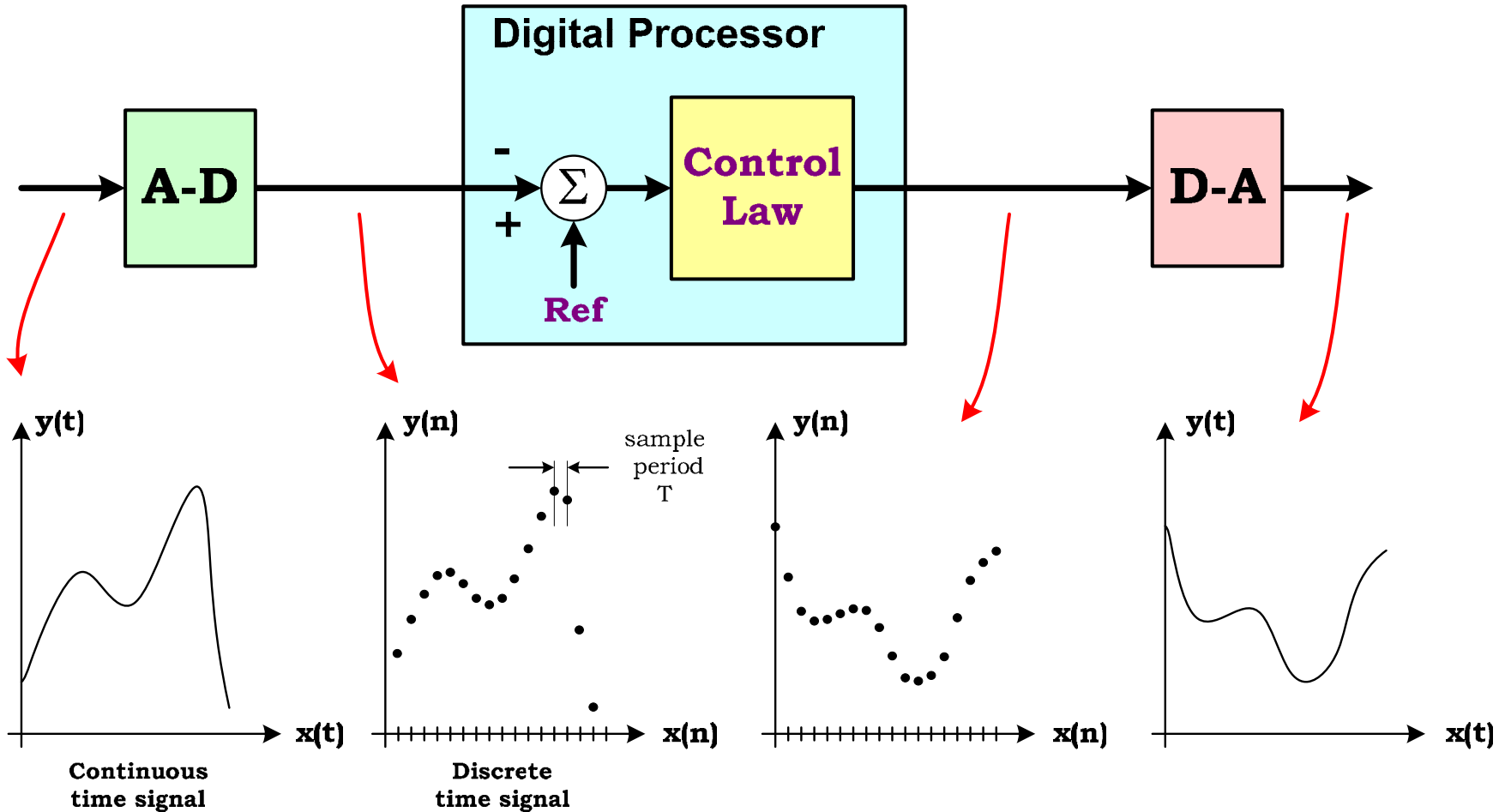
Core Loop/s

- Deterministic
- Real-Time dead lines always maintained
- Speed critical for high control bandwidth
- Synchronous to PWM and ADC timing

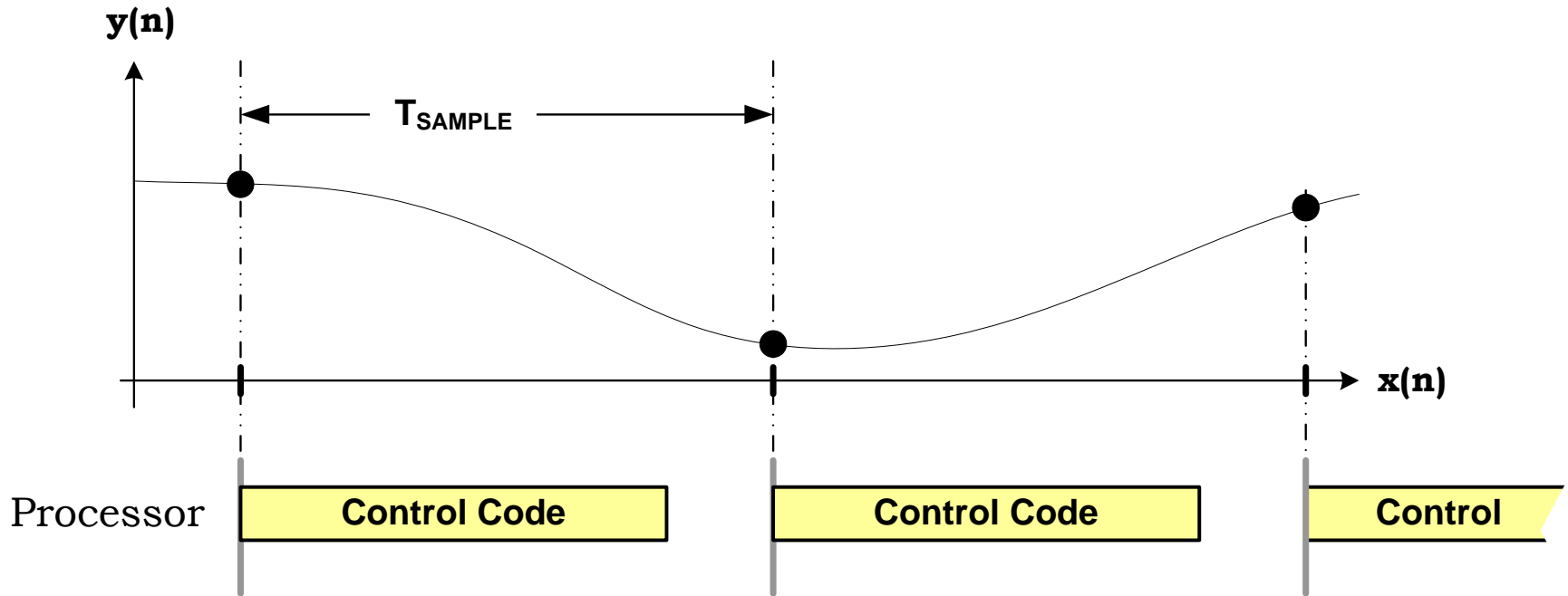
Loop management

- Soft-Start / Shutdown / Sequencing
- Duty clamping - e.g. under “unusual conditions”
- Fault Management
- Loop operating mode – coeff. selection
- Communications with Host

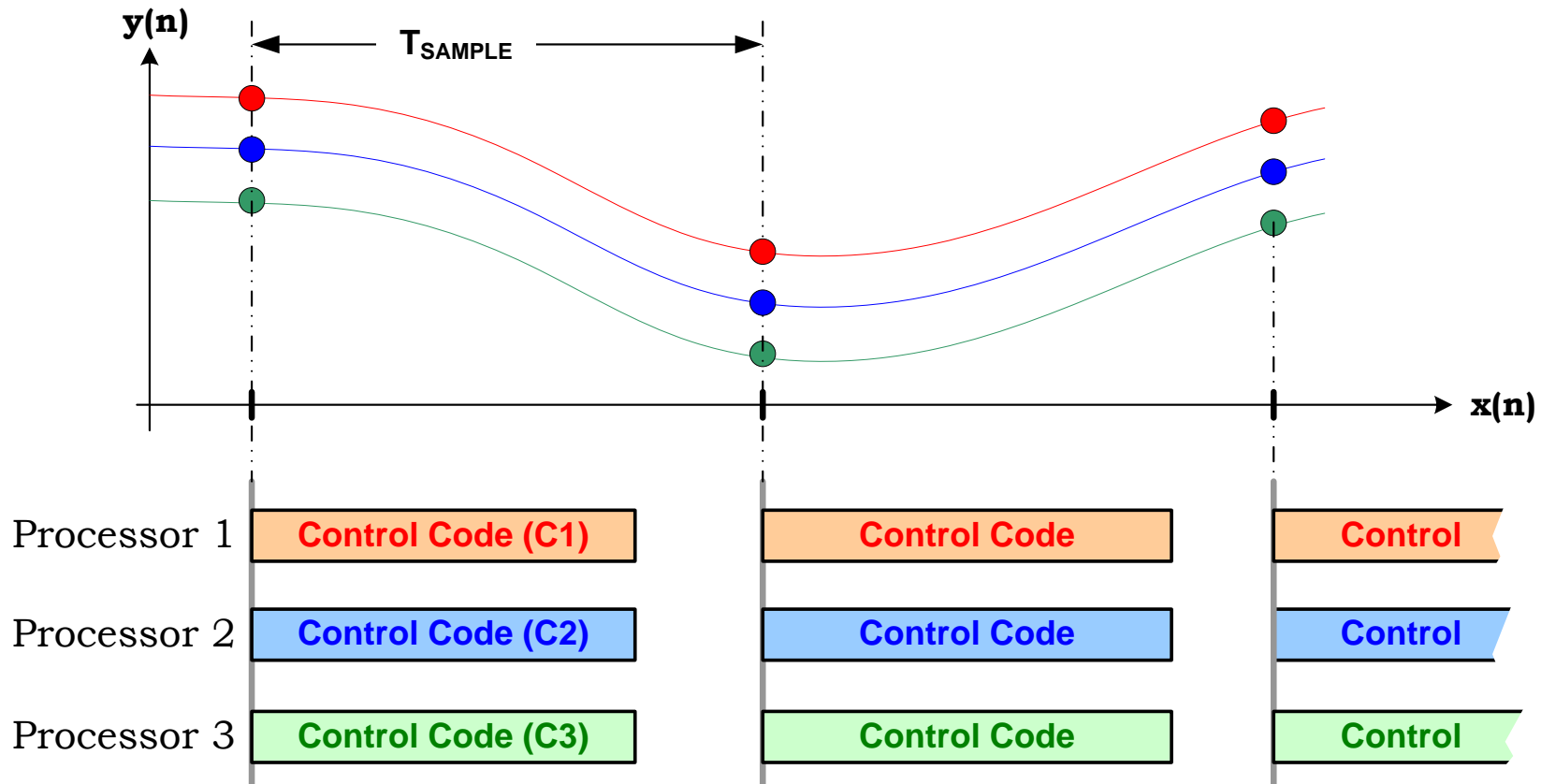
Time sampled systems



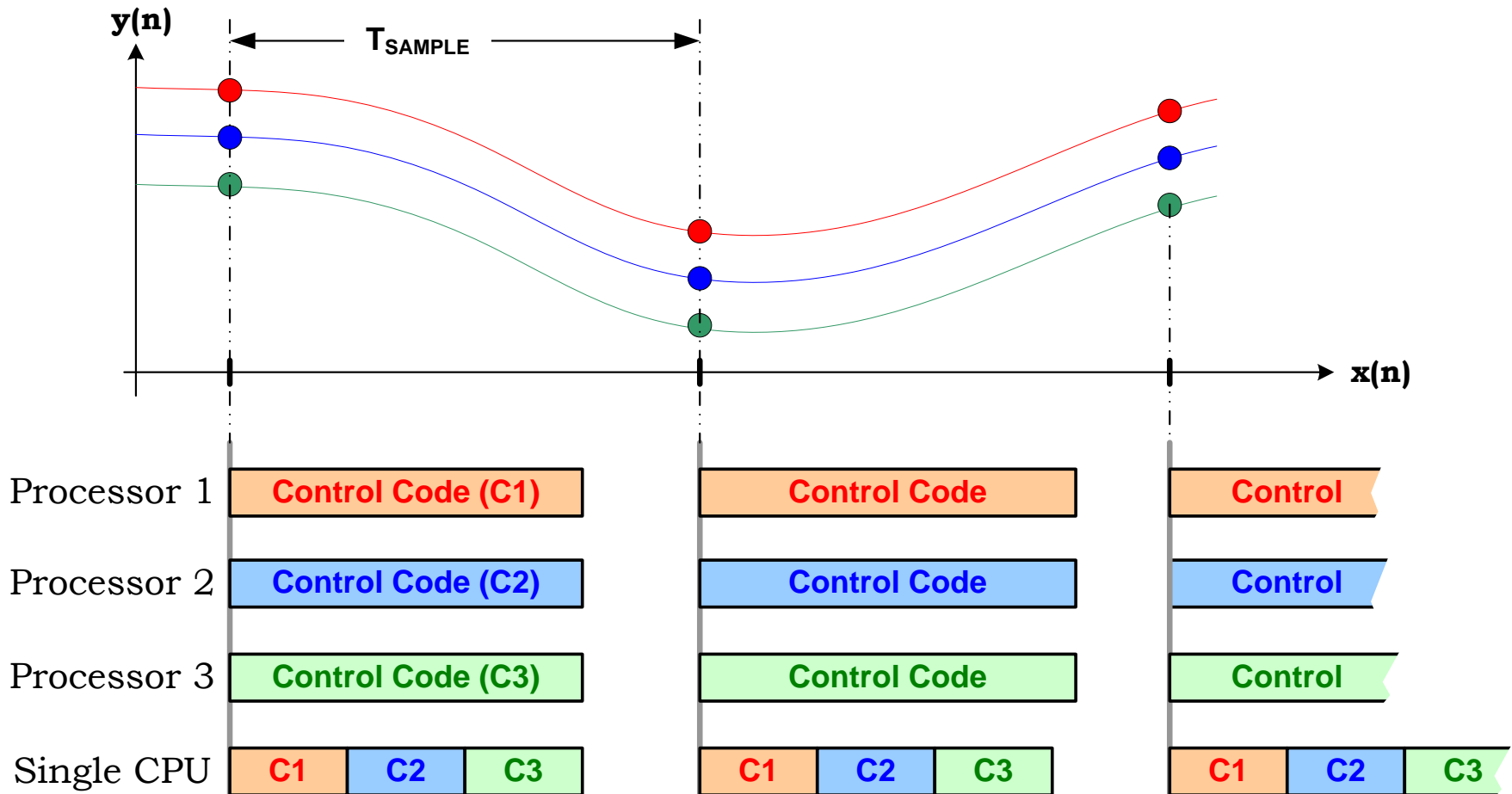
Time Division Multiplexing – TDM (1/2)



Time Division Multiplexing – TDM (2/2)



Time Division Multiplexing – TDM (2/2)



Software Framework for a Digital Controller

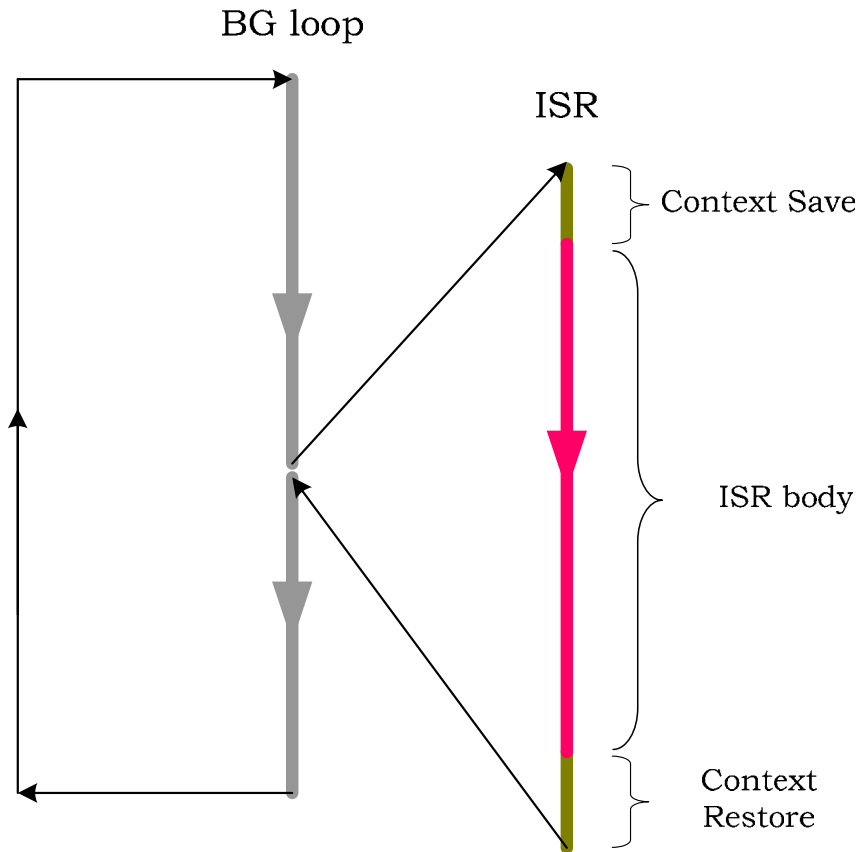
“infrastructure which supports the application”

Considerations

- How many ISRs (Interrupt Service Routines)
- Are ISRs Synchronous or Asynchronous ?
- CPU % utilization balance between ISRs and Background (BG)
- High level language (HLL), e.g. “C/C++”, Assembly ?, or both ?
- Need to employ an Operating system ?
- Interrupt driven Communications ?

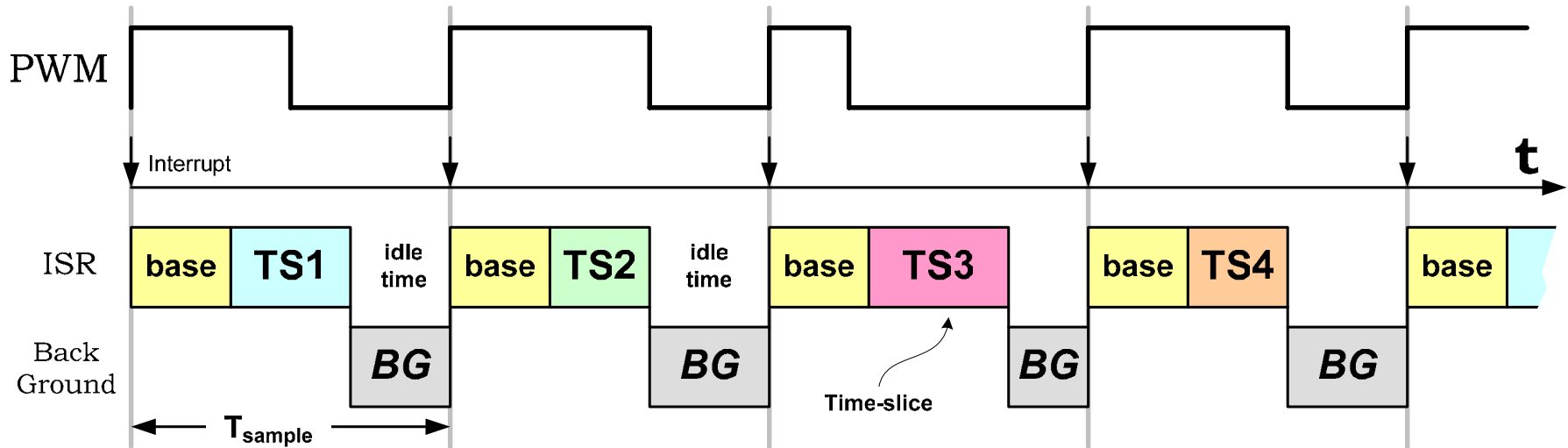
The simple “ISR / BG” Framework (1/2)

“keep it simple”



- **2 Loops only**
- **ISR code has highest priority**
- **ISR Synchronous to PWM switching**
- **ISR incurs entry/exit overhead**
- **BG runs only during ISR “idle time”**

The simple “ISR / BG” Framework (2/2)



- Can Time slice the ISR for simple synchronous multi-task scheduling
- In a practical system BG needs approx 15~20% of CPU bandwidth
- If CPU timing is “tight” may need to consider slowing the “loop”

CPU Performance requirements

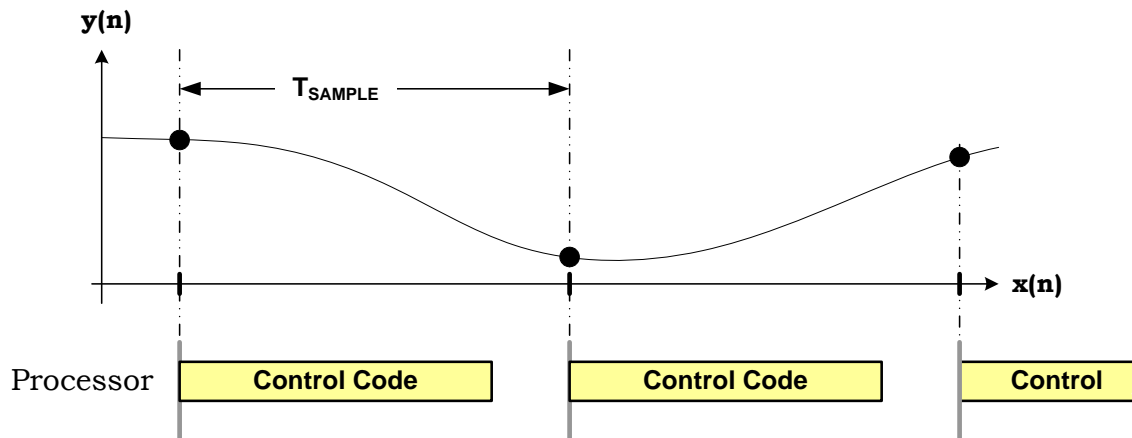
Key Considerations

- Available processing time
- MIPS (Millions of Instructions Per Second) available
- ISR bandwidth utilization
- BG bandwidth (average Background code bandwidth)

CPU Performance requirements (cont.)

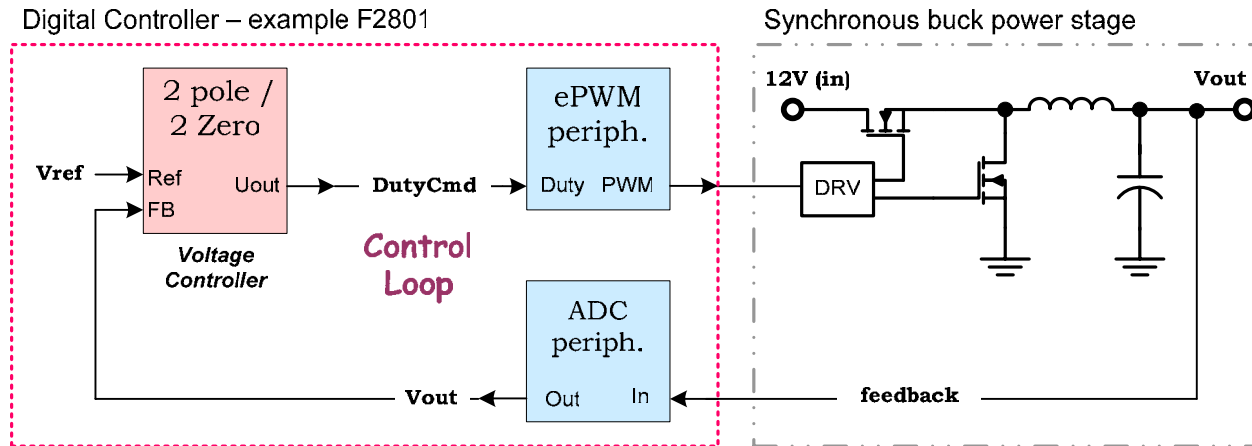
➤ Available processing time

Sample Freq (=PWM) (KHz)	Sample Period (nS)
100	10000
300	3333
500	2000
700	1429
1000	1000
1500	667
2000	500



CPU Performance requirements (cont.)

➤ ISR Bandwidth utilization



$$\frac{U(Z)}{E(Z)} = \frac{B_0 + B_1Z^{-1} + B_2Z^{-2}}{1 + A_1Z^{-1} + A_2Z^{-2}} \dots \dots \dots (1)$$

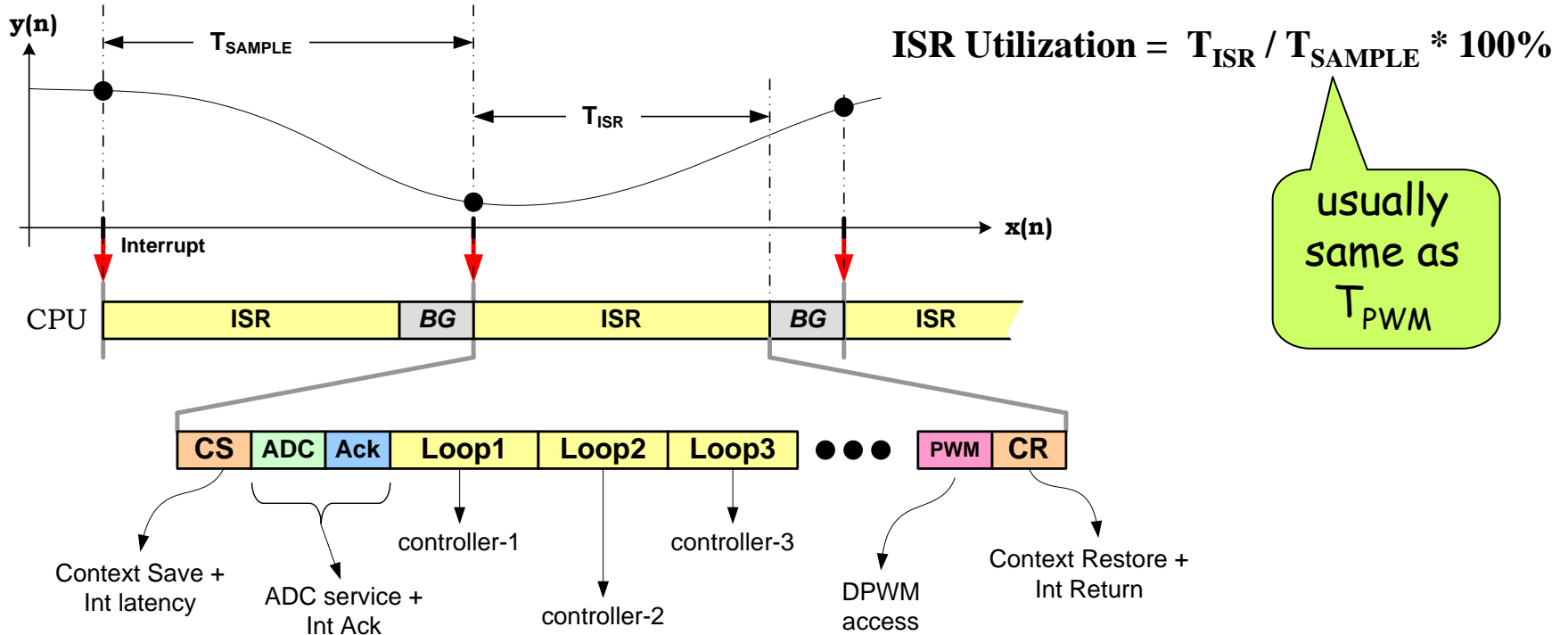
$$U(n) = B_0 \cdot E(n) + B_1 \cdot E(n-1) + B_2 \cdot E(n-2) + A_1 \cdot U(n-1) + A_2 \cdot U(n-2) \dots \dots \dots (2)$$

$$E(n) = V_{out} - V_{ref} \dots \dots \dots (3)$$

$$U_{Qo}(n) = PrdSF \cdot U(n) \dots \dots \dots (4)$$

CPU Performance requirements (cont.)

➤ ISR Bandwidth utilization (cont'd)



Operation	# Clock Cycles (1 loop)	# Clock Cycles (2 loops)	# Clock Cycles (3 loops)
Context Save + Int. latency	16	16	16
ADC servicing + Ack	4	5	6
2P/2Z controller	25	47	69
ePWM access	4	8	12
Context Restore + Int. Return	16	16	16
Total	65	92	119

CPU Performance requirements (cont.)

➤ ISR Bandwidth utilization (cont'd)

ISR Utilization for PWM frequency vs # Control loops

PWM		# LOOPS & # Cycles				
(KHz)	(uS)	1 65	2 92	3 119	4 146	5 173
200	5.00	13%	18%	24%	29%	35%
300	3.33	20%	28%	36%	44%	52%
400	2.50	26%	37%	48%	58%	69%
500	2.00	33%	46%	60%	73%	87%
600	1.67	39%	55%	71%	88%	104%
700	1.43	46%	64%	83%	102%	121%
800	1.25	52%	74%	95%	117%	138%
900	1.11	59%	83%	107%	131%	156%
1000	1.00	65%	92%	119%	146%	173%
1100	0.91	72%	101%	131%	161%	190%

Note: Entries in **red** require more than 100% and are not possible.

CPU Performance requirements (cont.)

➤ BG average bandwidth

- BG loop has access to CPU cycles remaining from ISR, i.e.
- BG BW = 100% - ISR utilization
- BG MIPS = % BG BW * CPU MIPS
- BG LR (loop rate) = BG MIPS / (# instructions in longest path)

BG Loop rate Example

Case: 600KHz, 4 loops, ISR utilization = 88%, longest s/w path = 300 inst.

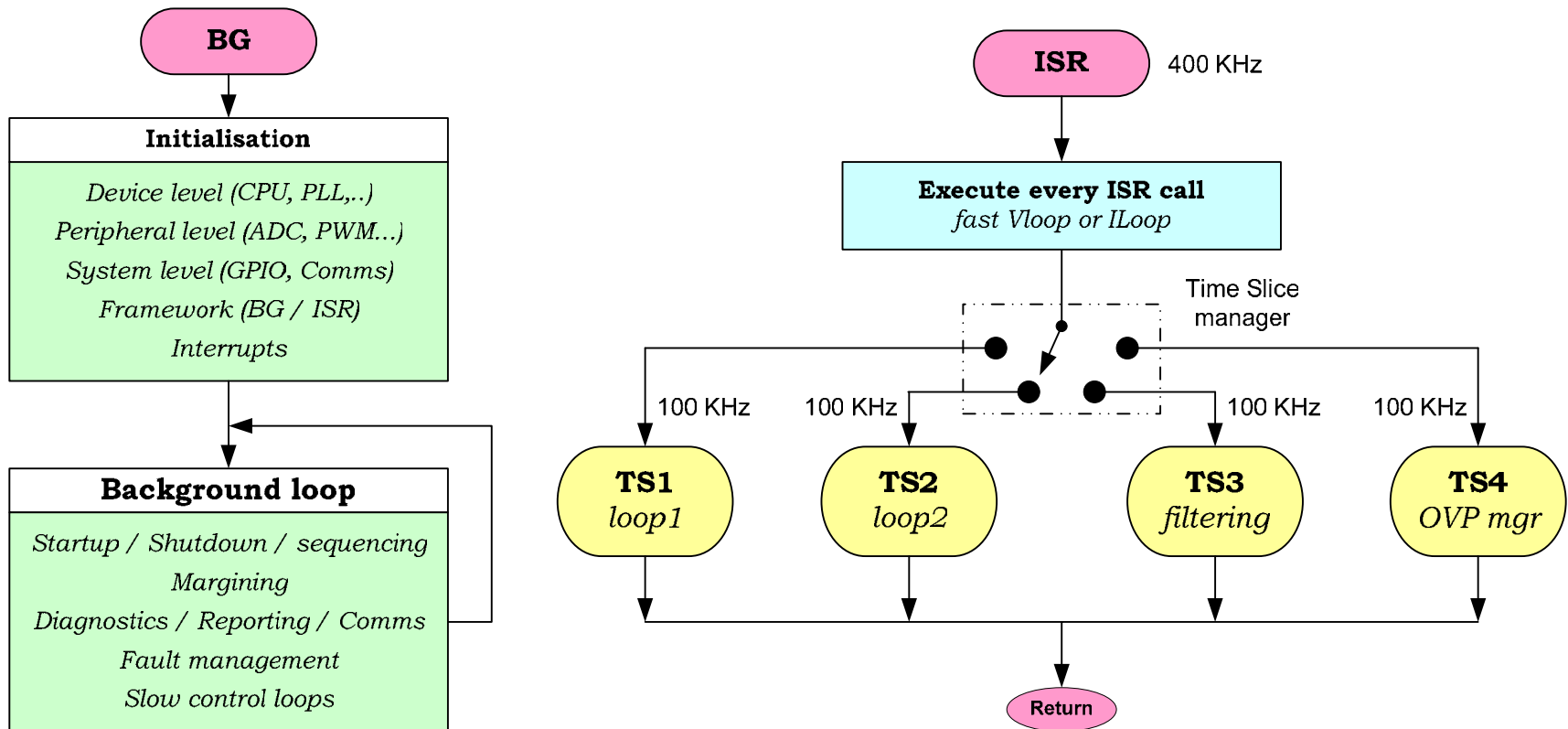
BG BW = 100% - 88% = 12%

BG MIPS = 12% * 100 MIPS = 12 MIPS

BG LR = 12,000,000 / 300 inst = 40 KHz

Practical Guidelines for ISR and BG

Single ISR / BG loop example



Common concerns with ASM s/w (1/2)

➤ Coding complexity, maintenance burden

- ISR code size bounded by “physics” → time deadlines (see table)
- Little or no conditional type structure, makes things simpler
- Mostly math / algorithm based, hence won't change very often
- Optimized Software function libraries available free from TI

Instructions (red) possible in a Sample period for a given CPU MIPS

F _{SAMPLE} (=PWM) (KHz)	MIPS		
	25	60	100
200	125	300	500
250	100	240	400
300	83	200	333
350	71	171	286
400	63	150	250
500	50	120	200

C code, ASM or mix of both ?

➤ **BG loop considerations**

- Not so time critical
- Contains ~90% of the total code
- Gives the application it's "Personality" or differentiation
- Can be quite complex, full of decisions.



Conclusion is easy: Use C or C++

➤ **ISR loop considerations**

- Can have huge impact on system S/W performance (bandwidth)
- Contains a very small portion of the total code (<10%)
- All things being equal, ASM can extract maximum performance from a CPU, could be a competitive advantage..."do more for less"
- In many cases, C / C++ is a valid choice for the ISR



Conclusion: if performance is key, use ASM

Common concerns with ASM s/w (2/2)

➤ Tangible benefit from using ASM ?

- If CPU loading (ISR utilization) is low, e.g. 20%, then benefit is low
- Conversely, benefit can be significant if utilization is close to 100%, e.g.

% impact (benefit) of 10 instructions on ISR utilization for PWM freq vs CPU MIPS

F _{SAMPLE} (=PWM) (KHz)	MIPS		
	25	60	100
200	8.0%	3.3%	2.0%
250	10.0%	4.2%	2.5%
300	12.0%	5.0%	3.0%
350	14.0%	5.8%	3.5%
400	16.0%	6.7%	4.0%
600	24.0%	10.0%	6.0%

$15/10 * 6\% = 9\%$

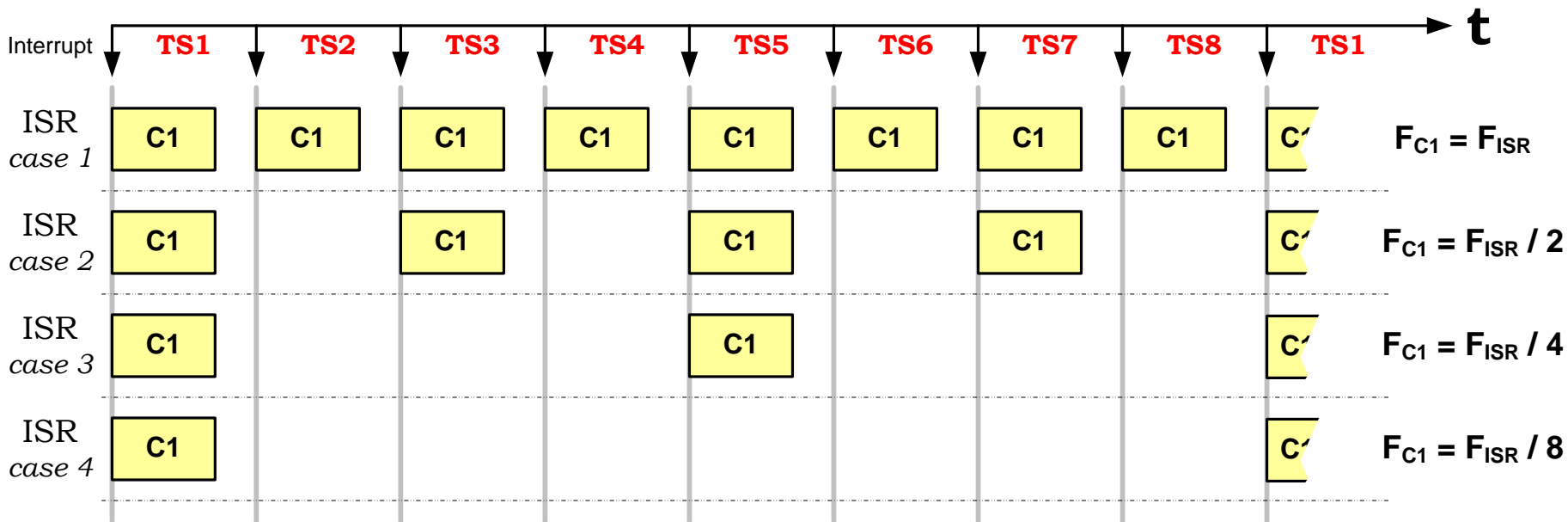
Recall example: BG LR = 12,000,000 / 300 = 40 KHz, based on ISR util. = 88%
Now assume a 9% boost is achieved by saving 15 instructions (i.e. 600KHz vs 100MIPS)
Now ISR util. = 88% - 9 = 79%, and BG BW = 100% - 79% = 21% (i.e. 21 MIPS)
BG LR = 21 MIPS / 300 inst. = 70 KHz ... a significant boost with just 15 instructions.

Loop control

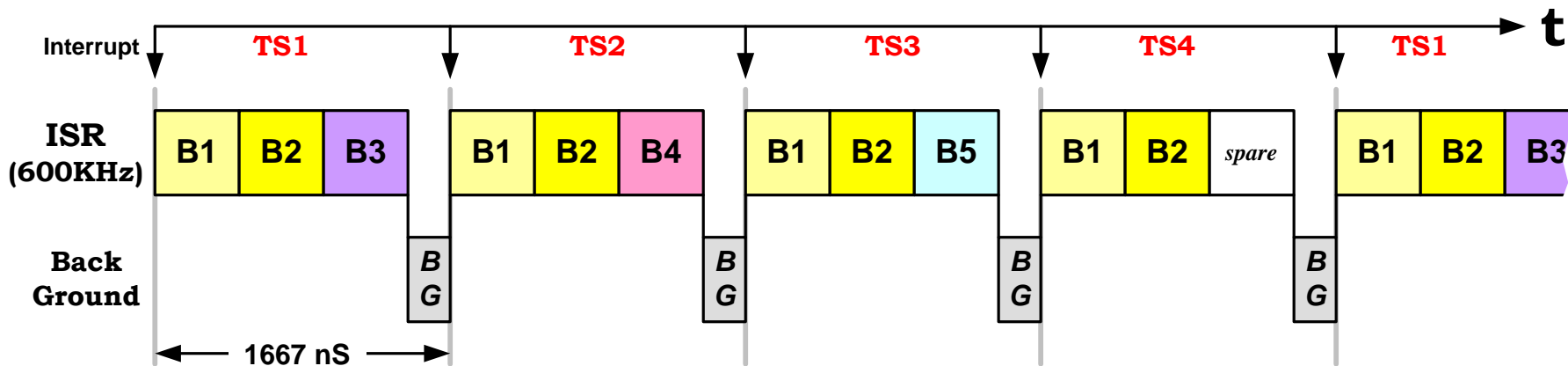
ISR

considerations

Time Sliced ISR – operating principle

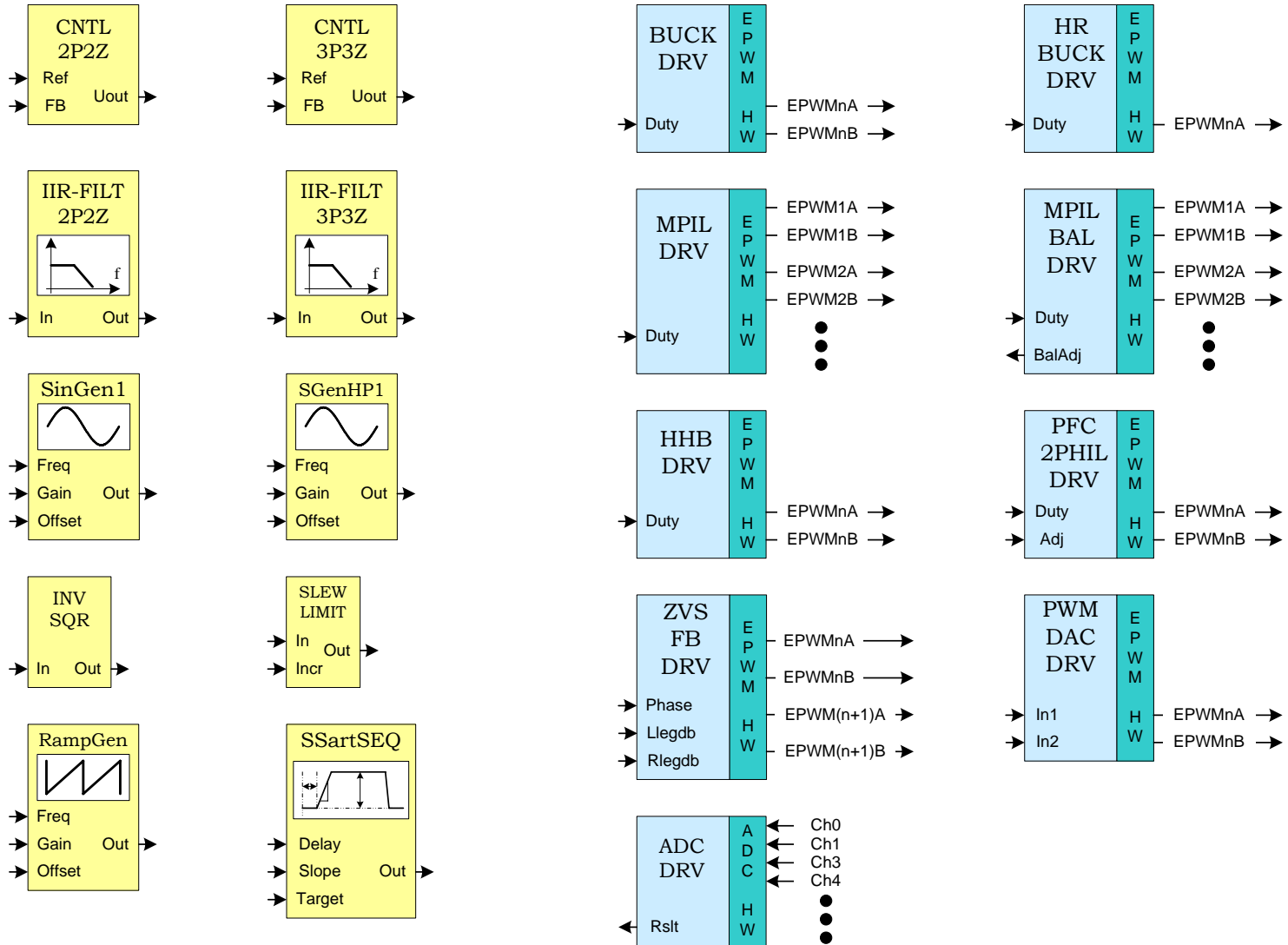


Time Sliced ISR – Practical example



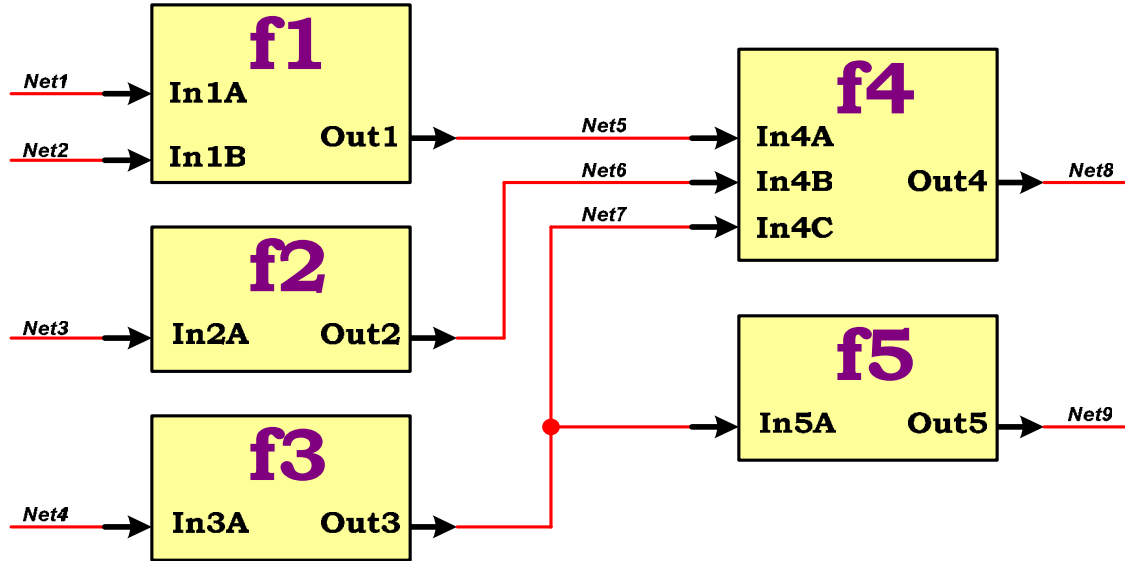
Code Function	PWM rate (KHz)	Code exec. rate (KHz)	Identifier
Buck 1 - single phase V Loop	600	600	B1
Buck 2 - single phase V Loop	600	600	B2
Buck 3 - 4-phase IL V Loop	300 / (90° phase)	150	B3
Buck 4 - 3-phase IL V Loop	300 / (120° phase)	150	B4
Buck 5 - 3-phase IL V Loop	300 / (120° phase)	150	B5

Software Library approach



Modular s/w architecture

“Signal Net” based module connectivity



Initialization time (“C” - BG)

```
// pointer & Net declarations
Int *In1A, *In1B, *Out1, *In2A,...
Int Net1, Net2, Net3, Net4,...

// “connect” the modules
In1A=&Net1; In1B=&Net2; Out1=&Net5;
In2A=&Net3; Out2=&Net6;
In3A=&Net4; Out3=&Net7;
In4A=&Net5; In4B=&Net6; In4C=&Net7; Out4=&Net8;
In5A=&Net7; Out5=&Net9;
```

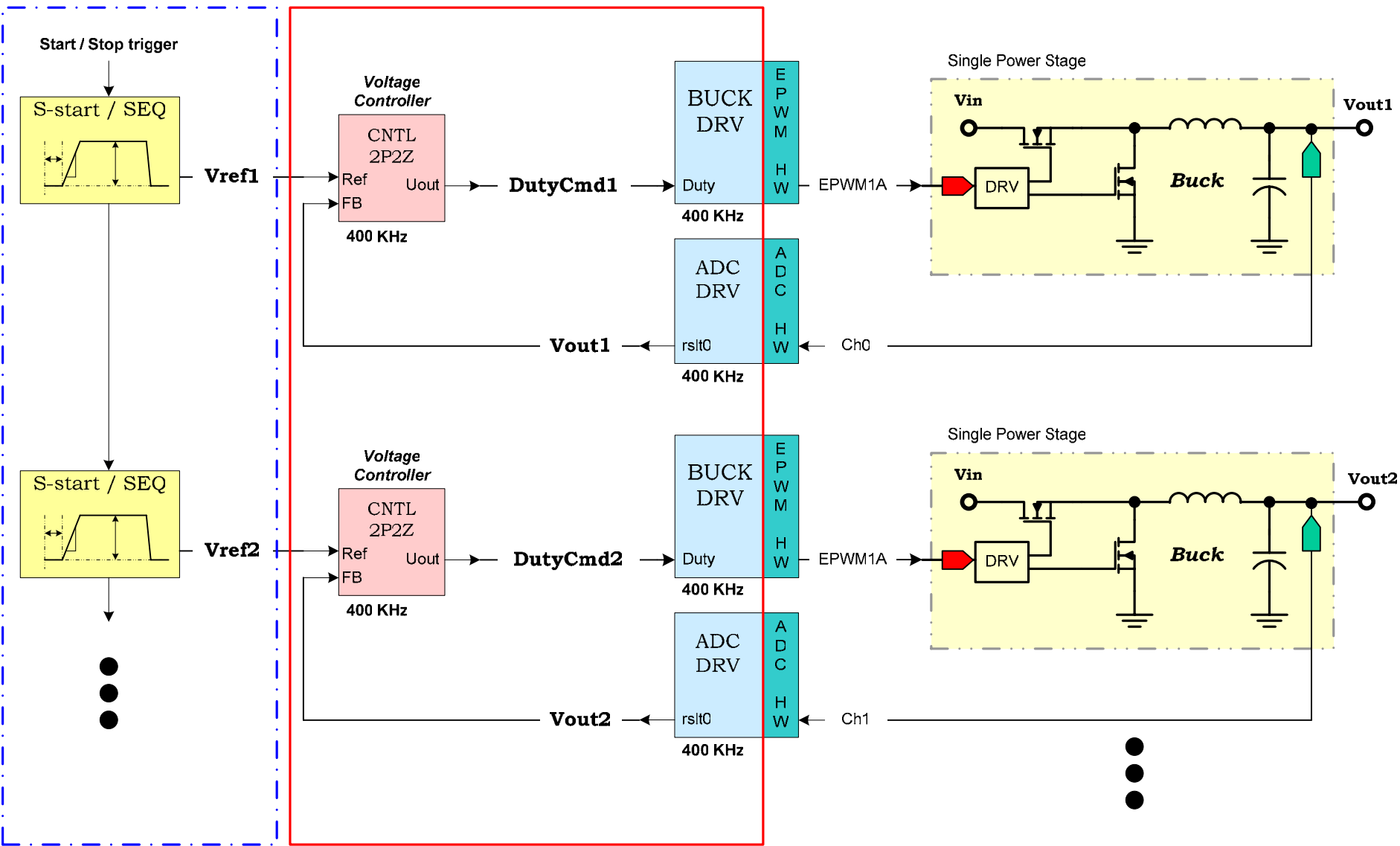
Run time - ISR ; Execute the code

```
f1
f2
f3
f4
f5
```

Multi-Output S/W management

BG

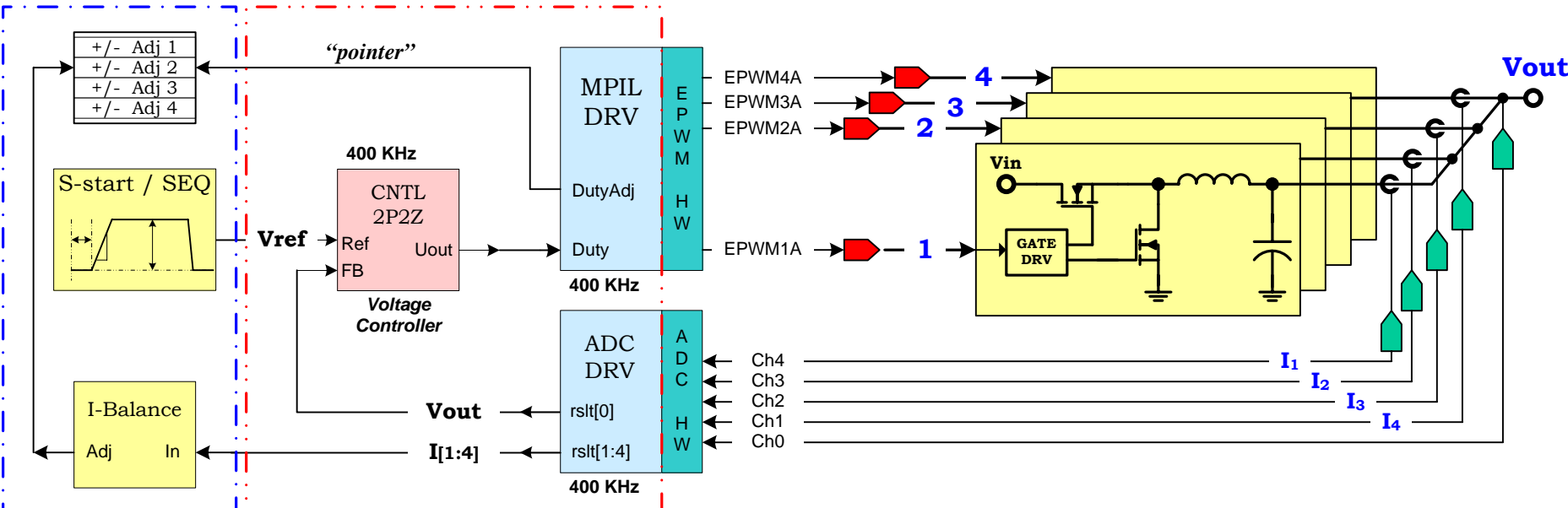
ISR



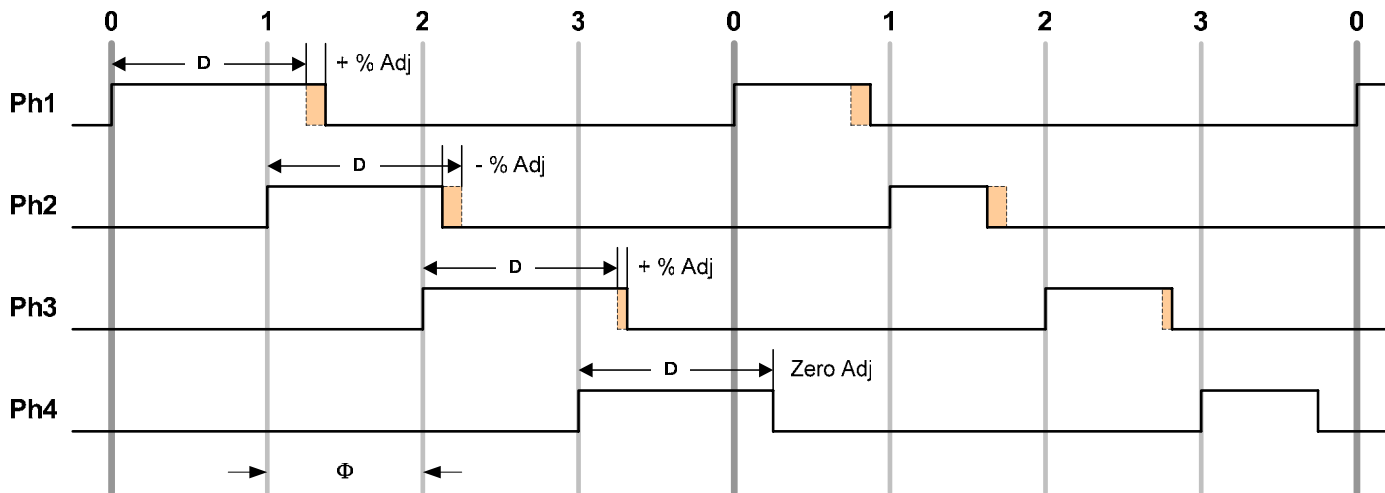
Multi-Phase IL S/W management

BG

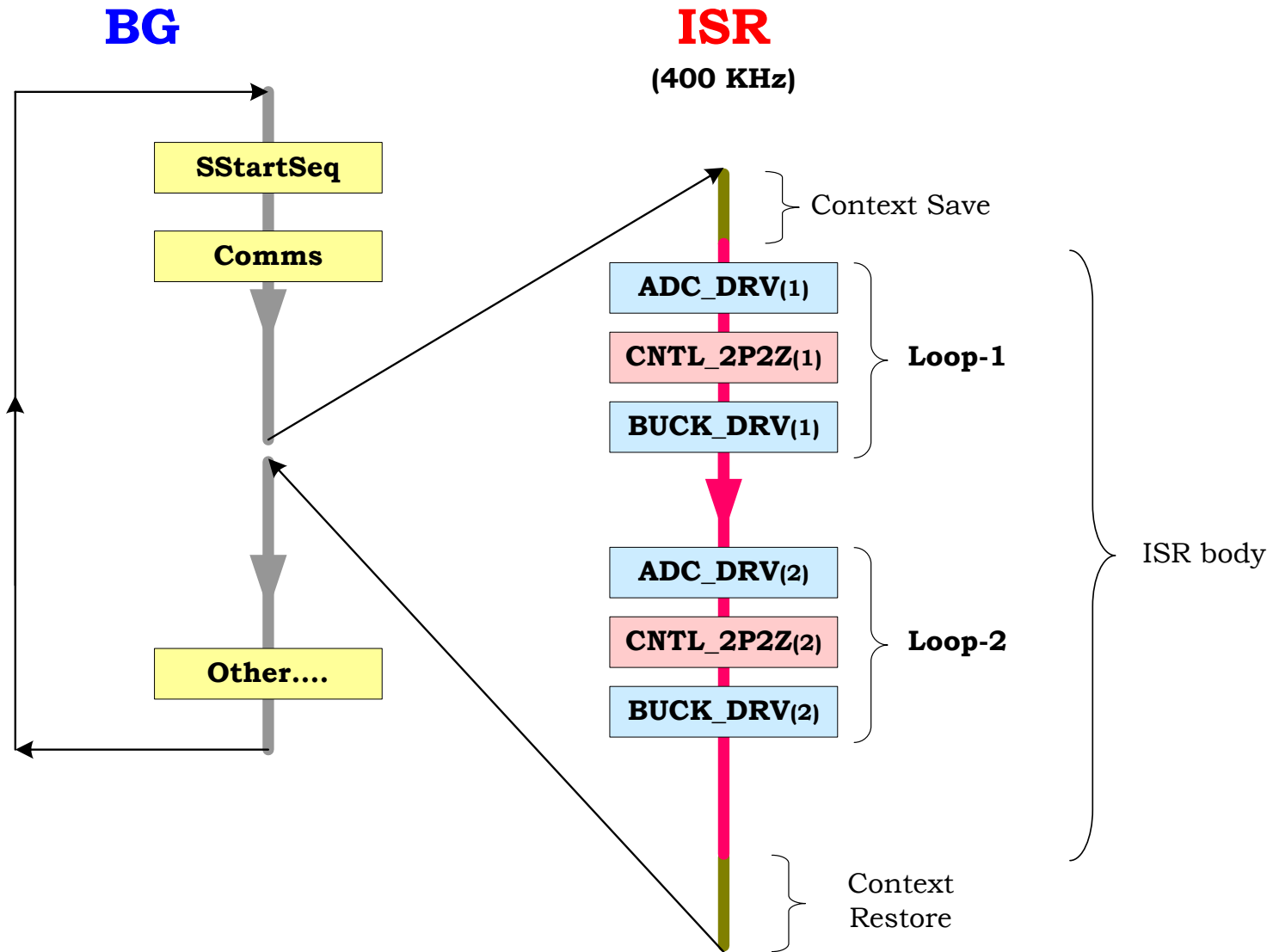
ISR



Phase Balancing with individual "duty adjust"



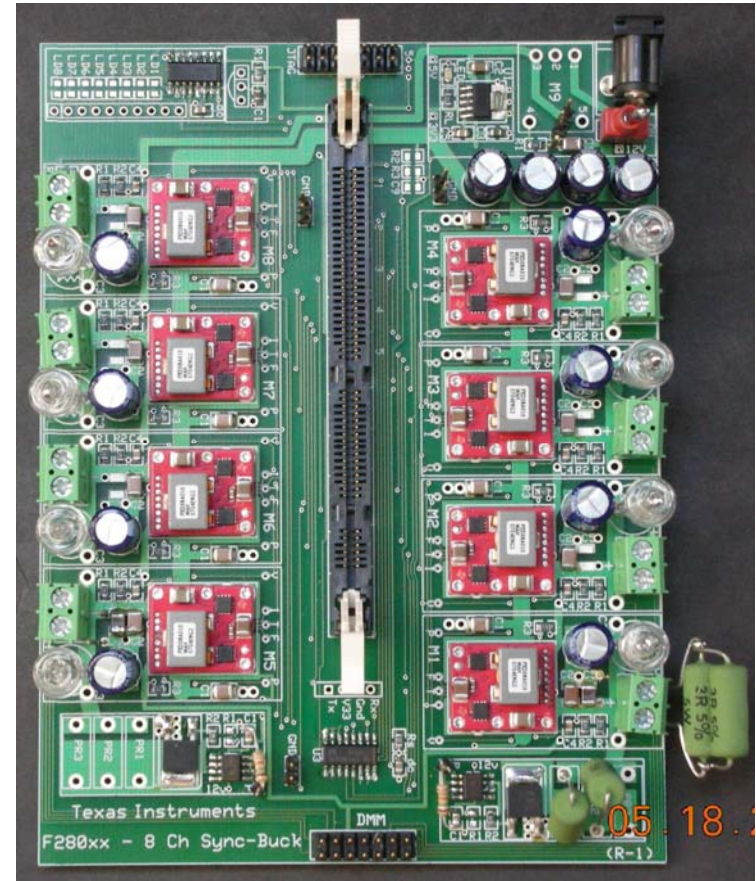
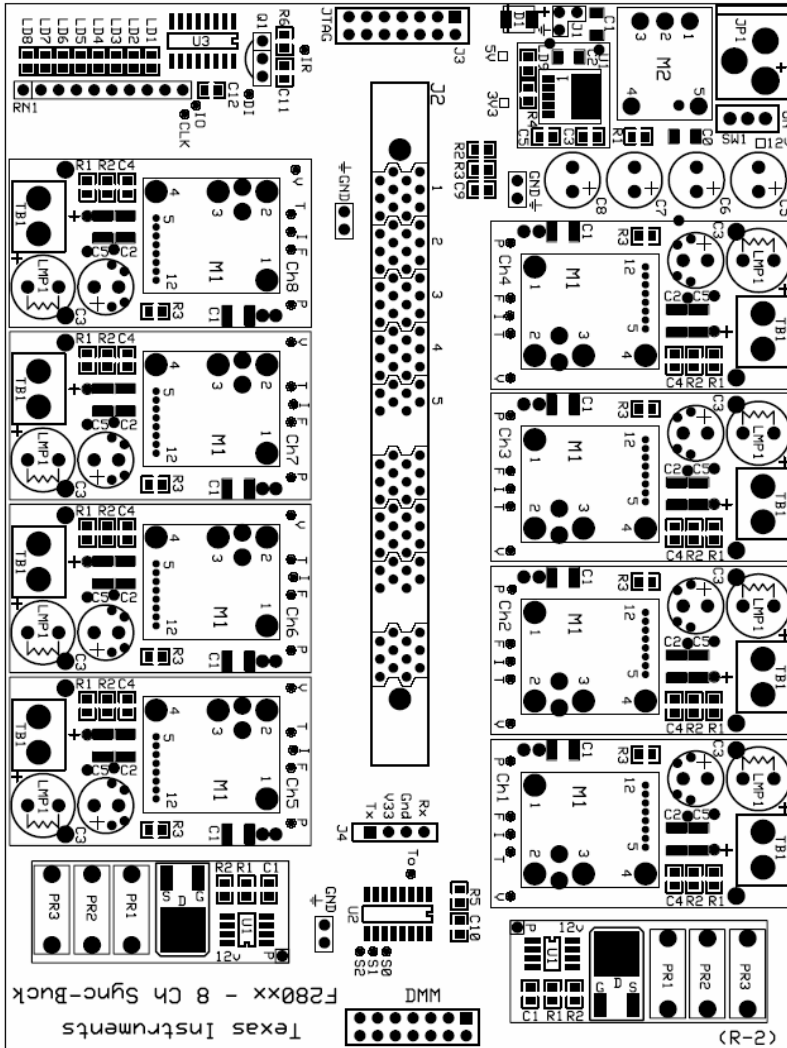
S/W block execution



Multi-Rail / Multi-Phase development tools

8 Rail / 8 Phase evaluation board

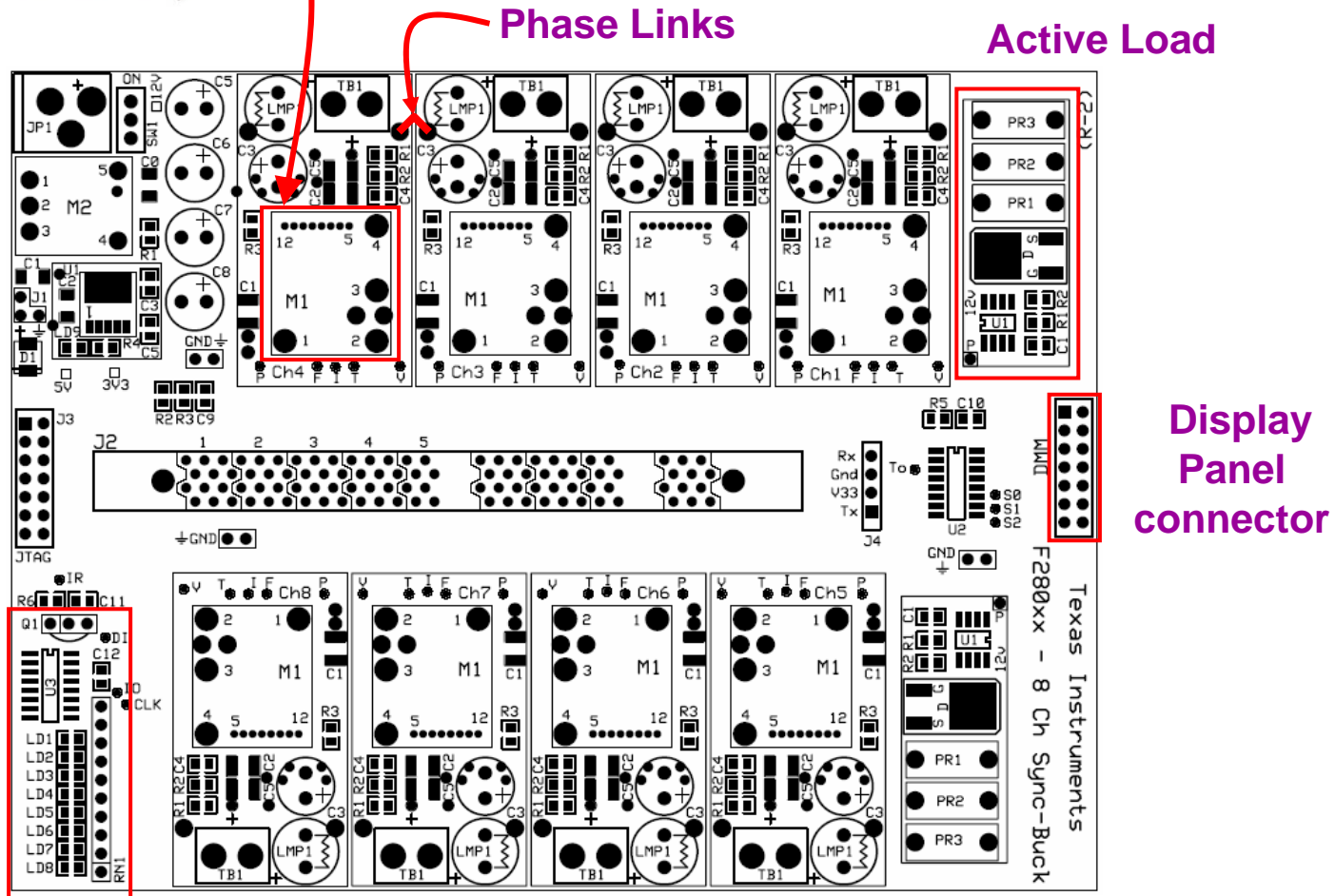
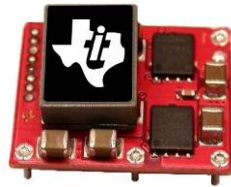
- 8 Channel proof-of-concept evm
- Multi-loop and sequencing
- Uses TI power stage modules



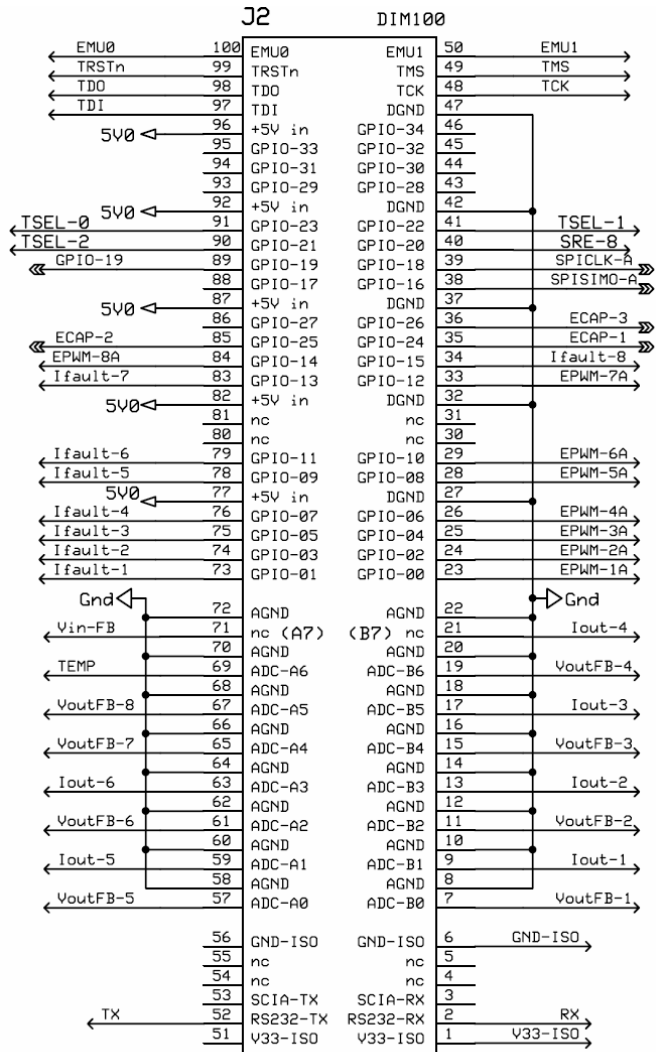
8 channel Buck EVM - features

PIP 10A module

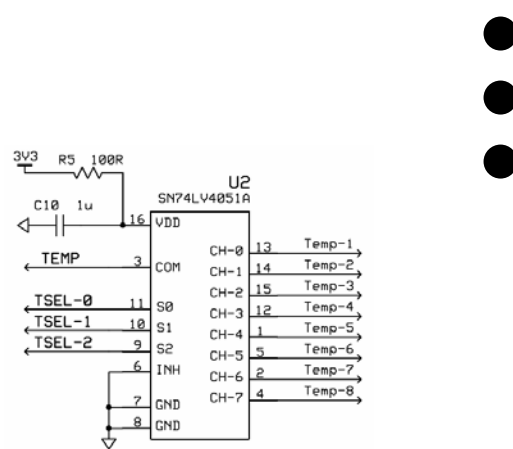
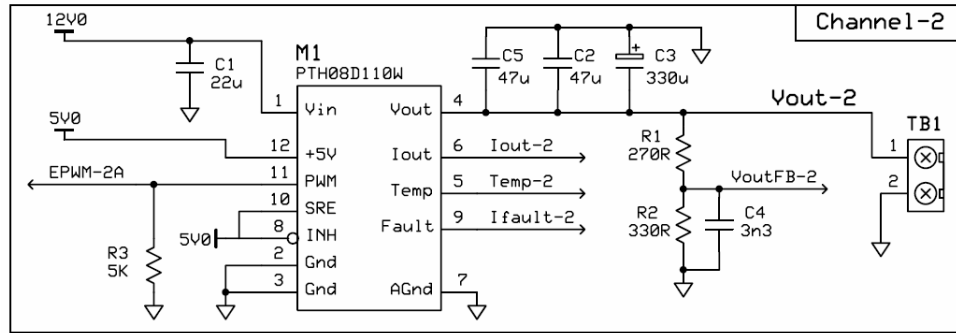
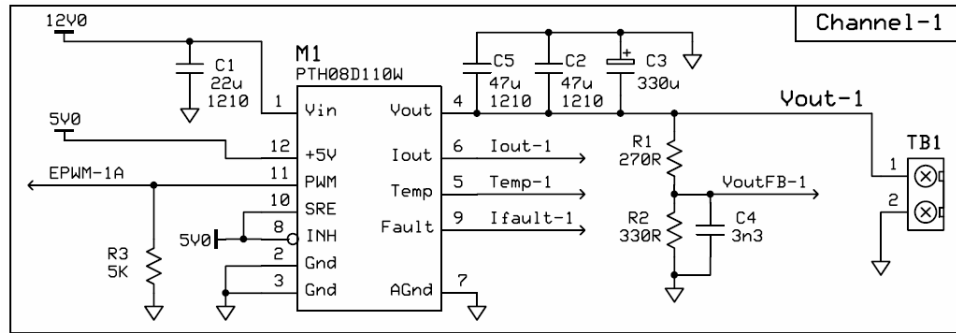
- Current meas.
- Temp meas
- Over Current Prot.
- Over Current Flag
- No Heat-sink needed



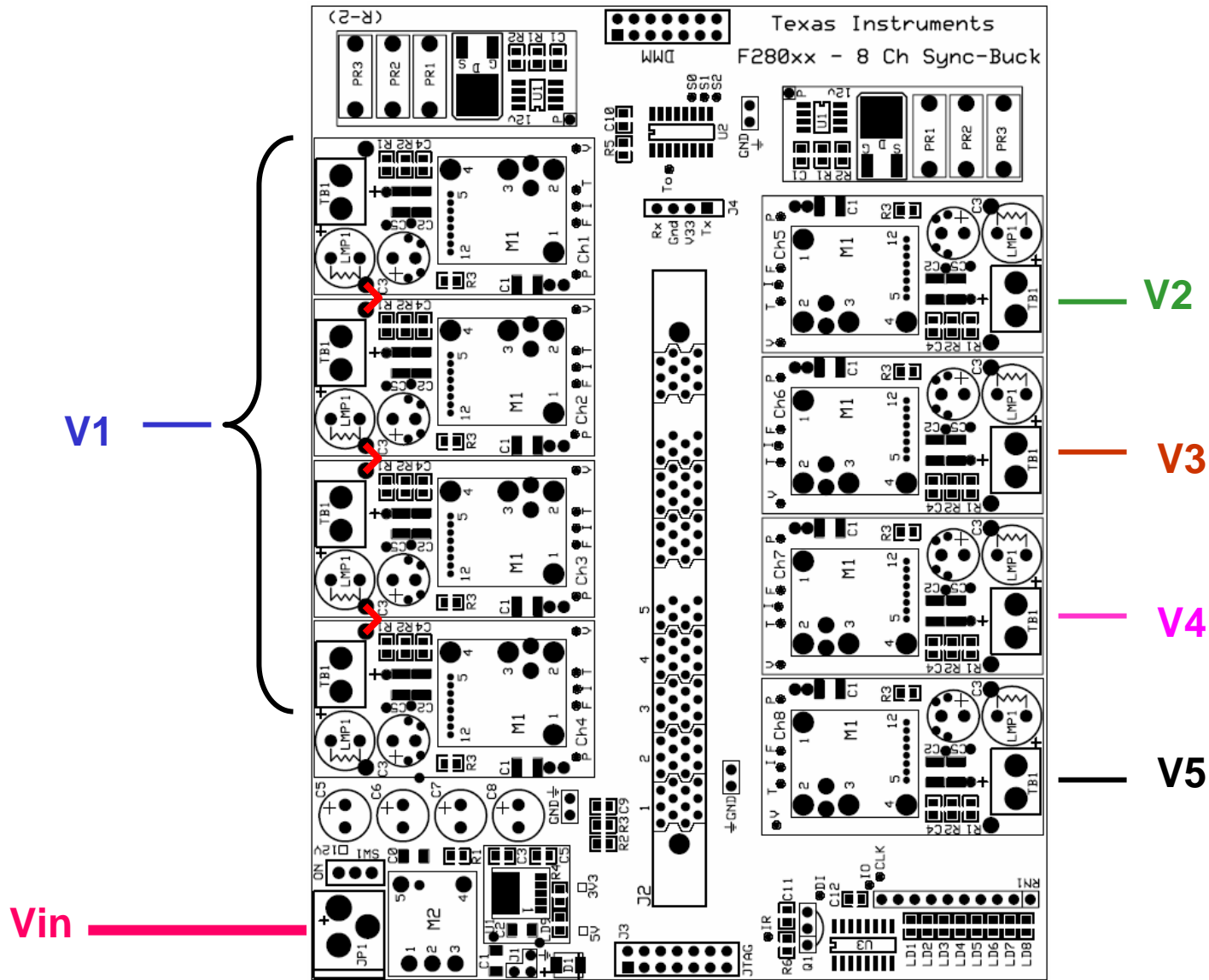
8 channel Buck EVM - schematic



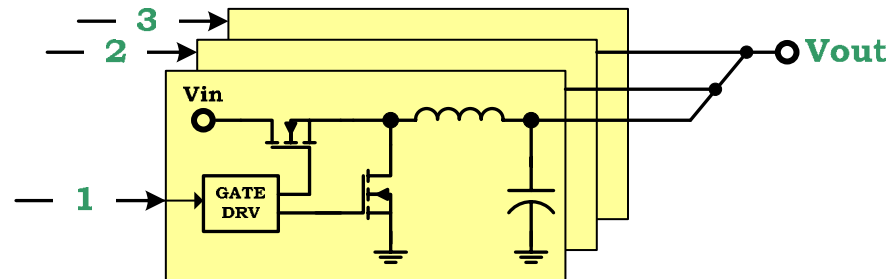
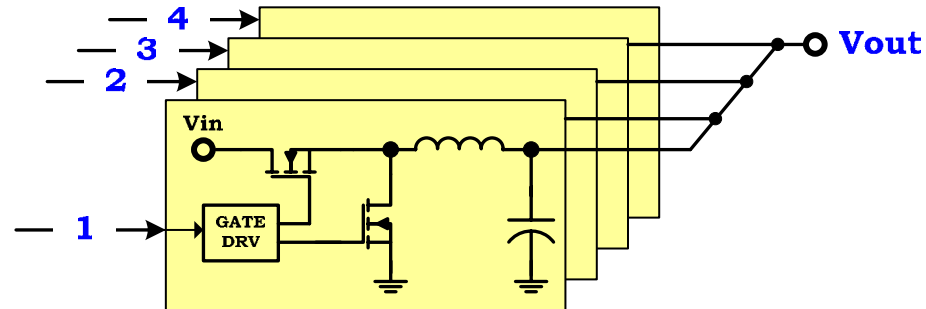
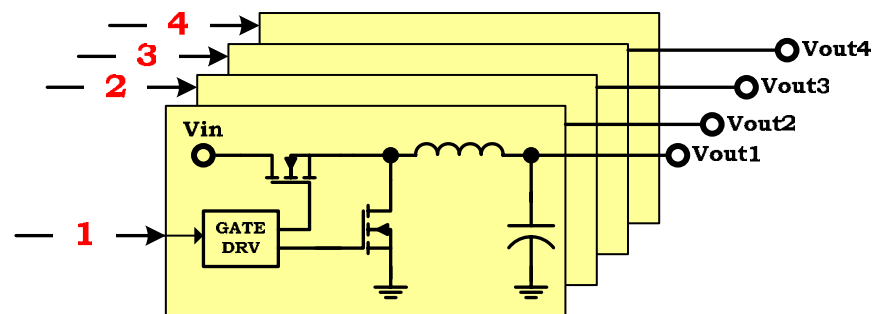
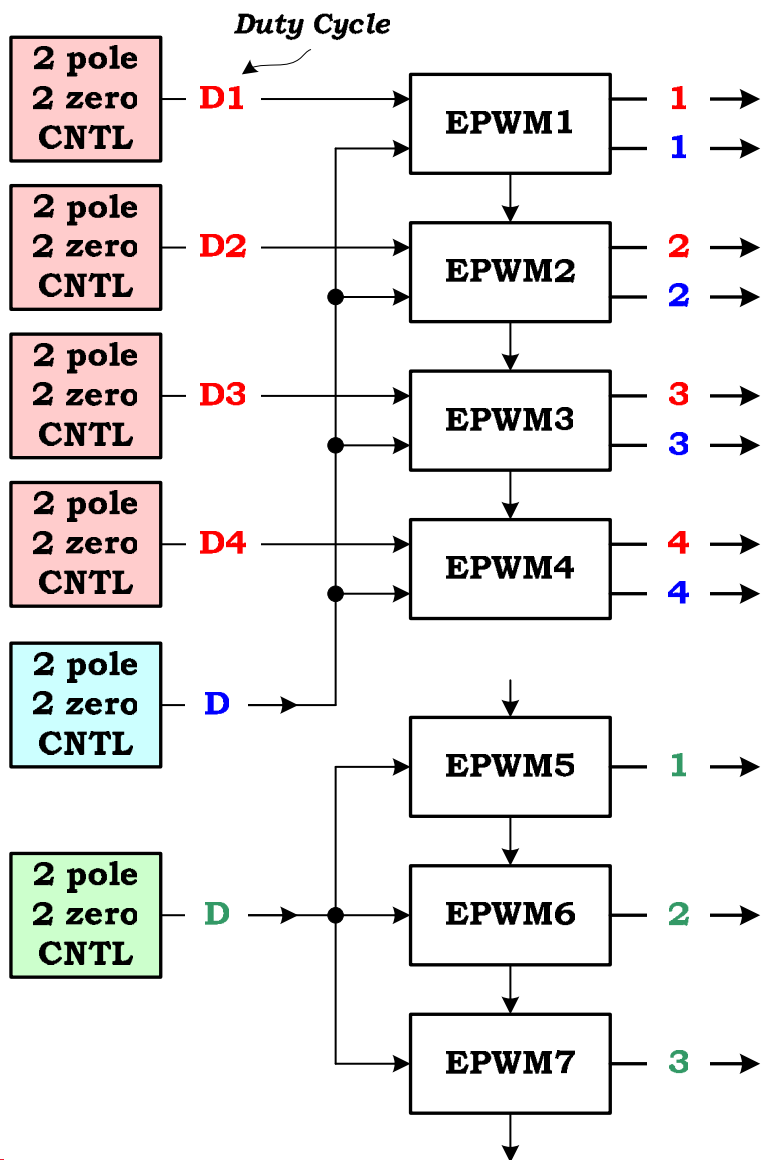
BH28044



example: Multi-Channel / Multi-Phase



Example DC/DC Combinations

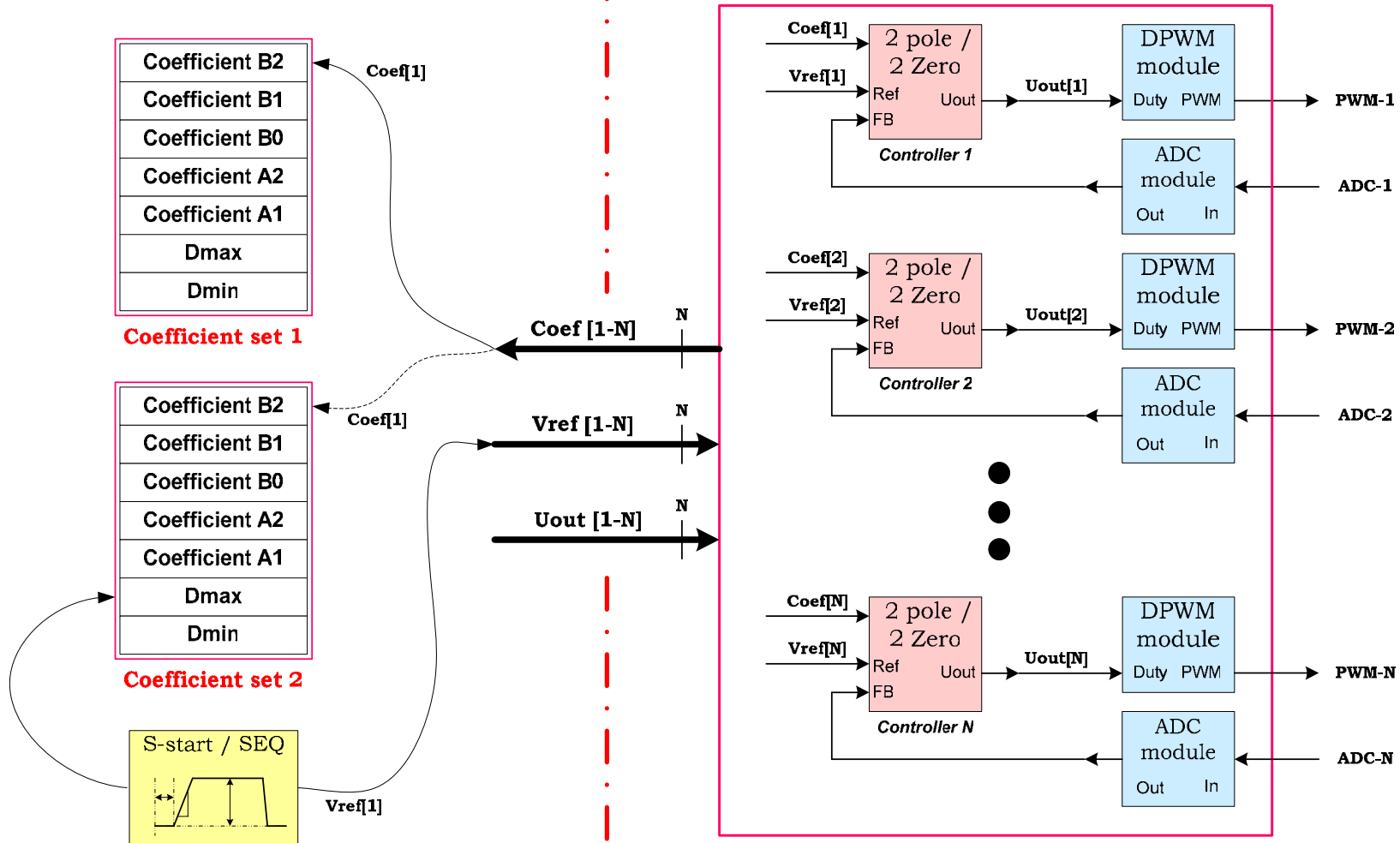


Multi-output control s/w module

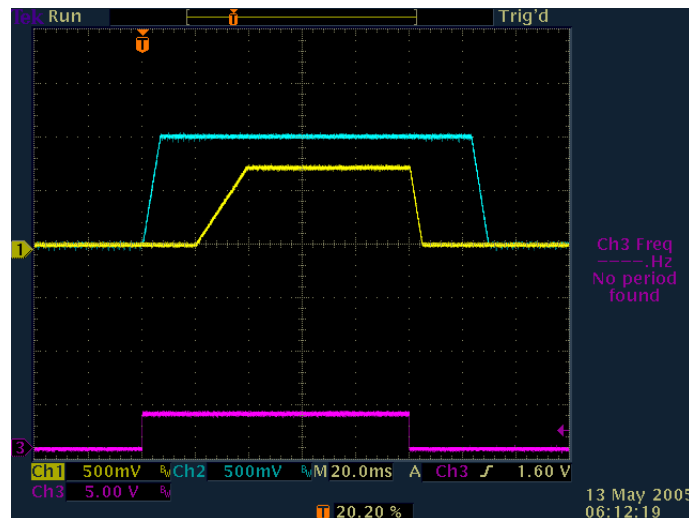
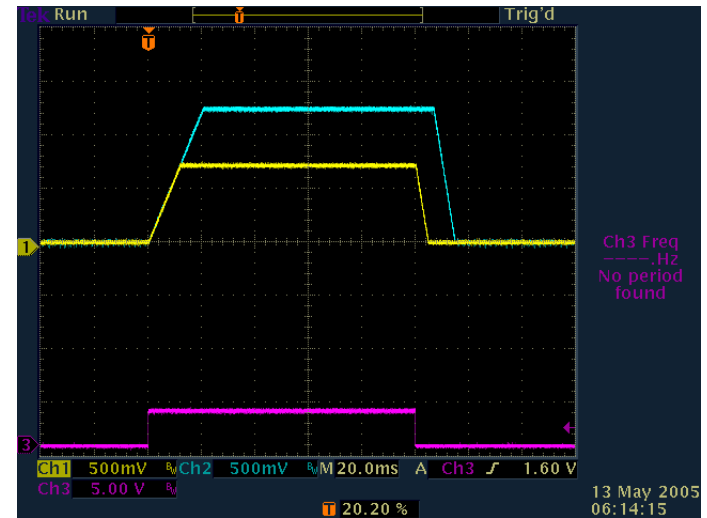
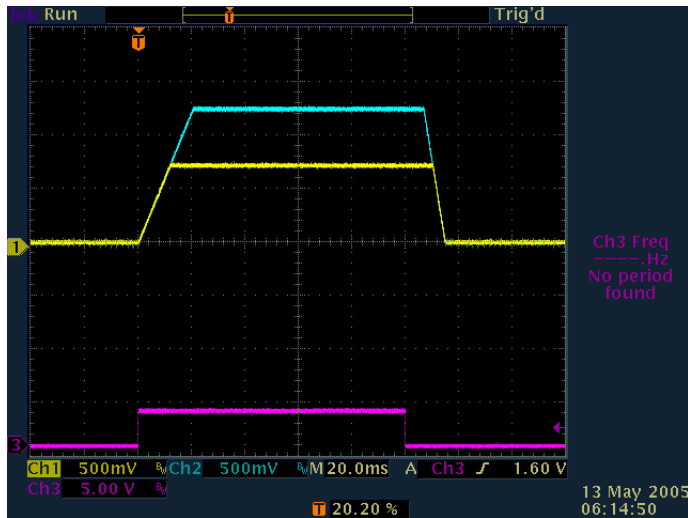
C / C++ (BG)

Assembly (ISR)

N-BuckLoop control module



Soft-start & Sequencing multi V_{out}



How many voltage Rails ?

CPU speed (MHz)	100	Control loop (cyc)		27	CPU prd (nS)	10.0	
Context save (cyc)	13	Misc Mgmt		4			
Context restore (cyc)	13	Background code (cyc)		300			
# DC/DC loops	200	250	300	400	500	1000	KHz
	5.0	4.0	3.3	2.5	2.0	1.0	uS
1	11.4%	14.3%	17.1%	22.8%	28.5%	57.0%	
BG spare (MIPs)	88.6	85.8	82.9	77.2	71.5	43.0	
BG loop spd (KHz)	295.3	285.8	276.3	257.3	238.3	143.3	
2	16.8%	21.0%	25.2%	33.6%	42.0%	84.0%	
BG spare (MIPs)	83.2	79.0	74.8	66.4	58.0	16.0	
BG loop spd (KHz)	277.3	263.3	249.3	221.3	193.3	53.3	
3	22.2%	27.8%	33.3%	44.4%	55.5%	111.0%	
BG spare (MIPs)	77.8	72.3	66.7	55.6	44.5	-11.0	
BG loop spd (KHz)	259.3	240.8	222.3	185.3	148.3	-36.7	
4	27.6%	34.5%	41.4%	55.2%	69.0%	138.0%	
BG spare (MIPs)	72.4	65.5	58.6	44.8	31.0	-38.0	
BG loop spd (KHz)	241.3	218.3	195.3	149.3	103.3	-126.7	
5	33.0%	41.3%	49.5%	66.0%	82.5%	165.0%	
BG spare (MIPs)	67.0	58.8	50.5	34.0	17.5	-65.0	
BG loop spd (KHz)	223.3	195.8	168.3	113.3	58.3	-216.7	
6	38.4%	48.0%	57.6%	76.8%	96.0%	192.0%	
BG spare (MIPs)	61.6	52.0	42.4	23.2	4.0	-92.0	
BG loop spd (KHz)	205.3	173.3	141.3	77.3	13.3	-306.7	
8	49.2%	61.5%	73.8%	98.4%	123.0%	246.0%	
BG spare (MIPs)	50.8	38.5	26.2	1.6	-23.0	-146.0	
BG loop spd (KHz)	169.3	128.3	87.3	5.3	-76.7	-486.7	
10	60.0%	75.0%	90.0%	120.0%	150.0%	300.0%	
BG spare (MIPs)	40.0	25.0	10.0	-20.0	-50.0	-200.0	
BG loop spd (KHz)	133.3	83.3	33.3	-66.7	-166.7	-666.7	